IPSJ Transactions on Advanced Computing Systems Vol.5 No.4 101–111 (Aug. 2012)

Regular Paper

Preventing Information Leakage from Virtual Machines' Memory in IaaS Clouds

Hidekazu Tadokoro[1,a)] Kenichi Kourai[2,4] Shigeru Chiba[3]

Received: January 18, 2012, Accepted: April 20, 2012

Abstract: Infrastructure as a Service (IaaS) provides virtual machines (VMs) to the users and its system administrators often manage the user VMs using privileged VMs called the *management VM*. However, the administrators are not always trustworthy from users' point of view. If the administrators allow outside attackers to intrude in the management VM, the attackers can easily steal sensitive information from user VMs' memory. In this paper, we propose *VMCrypt*, which preserves the data secrecy of VMs' memory using the trusted virtual machine monitor. VMCrypt provides a dual memory view: a *normal* view for a user VM and an *encrypted* view for the management VM. The encrypted view prevents sensitive information from leaking to the management VM. To support the existing management software for para-virtualization, VMCrypt exceptionally provides a normal view to the management VM only for several memory regions, which are automatically identified and maintained during the life cycle of a user VM. We have implemented VMCrypt in Xen and our experimental results show that the downtime due to live migration was still less than one second.

Keywords: virtual machine, IaaS cloud, information leakage, memory encryption

1. Introduction

Cloud computing, providing services via the Internet, is being widely accepted. Among various types of services, Infrastructure as a Service (IaaS) like Amazon EC2 [1] provides users with virtual machines (VMs), which are hosted in data centers. The users can migrate hardware, software, and data that they have possessed to IaaS clouds and use the necessary amount of VMs on demand. In IaaS, the users manage their VMs from remote sites via the Internet while the system administrators in IaaS manage the user VMs using privileged VMs called the *management VM*. For example, they may migrate user VMs to other hosts in clouds when necessary for load balancing or power saving.

Due to the privileges of the management VM, sensitive information in the user VMs may leak via the management VM. From users' point of view, the administrators in IaaS are not always trustworthy [2], [3], [4]. Lazy administrators may allow outside attackers to intrude in the management VM. Worse, administrators themselves may be malicious and perform insider attacks. Such attackers can easily steal sensitive information from the user VMs. However, it is difficult to prohibit the access to VMs' memory for the management VM because the administrators have to manage the user VMs, e.g., VM migration, by accessing their memory. This inherently leads to information leakage from the user VMs.

This paper proposes *VMCrypt*, which is a system that preserves the data secrecy of the user VMs' memory. VMCrypt provides a dual memory view: a normal view and an encrypted view. A user VM can run normally with a *normal* view whereas the management VM cannot steal information through an *encrypted* view. These two views coexist so that both VMs concurrently access the user VMs' memory. To allow the management VM to inspect memory regions of the user VMs for VM management, VMCrypt exceptionally gives a normal view to the management VM only for several regions. It automatically identifies such regions by monitoring the interaction between the management VM, the user VMs, and the virtual machine monitor (VMM). The information on the identified regions is cached during the life cycle of a user VM to preserve the compatibility with the existing management software.

We have implemented VMCrypt in Xen [5] and VMCrypt currently supports para-virtualized Linux as guest operating systems. Our experimental results show that VMCrypt allows the administrators to manage the user VMs securely and correctly. With VMCrypt, the administrators in the management VM failed to find cryptographic keys and passwords from the user VM's memory. Nevertheless, they could boot, suspend, resume, and migrate user VMs as usual. They could perform even live migration [6], [7] for running user VMs. The execution performance of VM management was degraded mainly by the overheads of cryptographic operations. However, the downtime due to live migration was still less than one second and its overhead was 13%.

The main contributions of this paper are as follows:

- Enabling live migration while the user VMs' memory is protected

[1] Tokyo Institute of Technology, Meguro, Tokyo 152–8552, Japan
[2] Kyushu Institute of Technology, Iizuka, Fukuoka 820–8502, Japan
[3] The University of Tokyo, Chiyoda, Tokyo 101–0021, Japan
[4] Japan Science and Technology Agency, CREST, Chiyoda, Tokyo 102–0076, Japan
[a)] tadokoro@csg.is.titech.ac.jp

- Supporting para-virtualized guest operating systems, which share various memory regions with the management VM
- Supporting the existing software for VM management

Several previous studies [3], [4], [8] tackle the problem of information leakage from the user VMs' memory, but none of them supports live migration. To the best of our knowledge, VMCrypt is the first system that supports the existing management software for para-virtualization.

The rest of this paper is organized as follows. Section 2 clarifies administrator-related information leakage in IaaS clouds and discusses related work. Section 3 presents VMCrypt to prevent information leakage from the user VMs. Section 4 describes the implementation details in Xen. Section 5 shows our experimental results for VM management with VMCrypt. Section 6 concludes the paper.

2. Motivation

2.1 Administrator-related Information Leakage

There are two types of system administrators in IaaS clouds. One is *IaaS users*, who are the administrators of the systems inside the VMs, called *user VMs*, in IaaS. The other is *IaaS administrators*, who manage virtualized systems themselves including the user VMs in IaaS. They can suspend, resume, and migrate user VMs as necessary. For such VM management, the IaaS administrators use privileged VMs called the *management VM*. The examples are domain 0 in Xen [5] and the service console in VMware ESX [9]. With the management VMs, the IaaS administrators can access the memory, disks, and networks of user VMs.

However, the IaaS administrators are not always trustworthy from IaaS users' point of view [2], [3], [4]. It is not rare that one cloud consists of several data centers around the world. The users often cannot know which data centers their VMs run in because the VMs may be migrated between data centers. The users cannot exactly know who are the administrators of their VMs. If a *lazy* administrator manages virtualized systems in IaaS, some management VMs may have vulnerabilities in software or configurations. In this case, vulnerable management VMs may be penetrated by outside attackers. If an administrator cannot fix those vulnerabilities for some reasons such as no patches, he is considered lazy as a result. If an IaaS administrator himself is *malicious*, he can perform insider attacks with high privileges. We refer to both outside attackers and malicious IaaS administrators simply as attackers.

In this situation, sensitive information in user VMs can leak via the management VM. Using the abilities of the management VM, the attackers inside it can steal the whole contents of the memory, disks, and networks of the user VMs on the same physical machine. For example, the attackers can dump the whole memory of the user VMs into their disks. They can mount the disks of the user VMs and read all the files. Network sniffers running in the management VM can easily capture packets to/from the user VMs. Fortunately, disks and networks could be encrypted by the guest operating systems themselves in the user VMs to protect their contents. The user VMs can use encrypted file systems like Windows EFS and virtual private networks (VPNs).

Unlike disks and networks, memory is crucial because it is difficult for the guest operating systems to encrypt their own memory. Without hardware support, neither operating systems nor applications can run with encrypted memory. There are various types of sensitive information in memory [10]. Clear-text passwords and cryptographic keys can be extracted from the user VMs' memory. Such information resides in buffers in the kernel or processes temporarily. In addition, the memory usually contains the buffer cache, which is used for maintaining copies of the data of file systems in memory. Through the analysis of the buffer cache, the attackers can read the data blocks of specific files if the blocks are on the buffer cache. The encryption of the file systems in the user VMs is not sufficient because the buffer cache cannot be encrypted.

Therefore, the confidentiality of the user VMs' memory should be preserved while the IaaS administrators perform VM management. A simple solution to prevent information leakage is to disable the management functions of the management VM, but this would not be acceptable in IaaS. At least, live migration is indispensable for the reasons of load balancing and power saving with negligible downtime. In particular, it is challenging to support para-virtualized guest operating systems because the management VM has to inspect and modify the user VMs' memory. Although full virtualization is being used, para-virtualization is still useful in terms of efficiency. In addition, any modification to the existing management software should not be required because it is not realistic to modify various management software.

2.2 Related Work

There are several studies for preventing information leakage via the management VM. CloudVisor [4] runs a security monitor underneath the VMM using nested virtualization. It encrypts all the memory pages of the user VMs whenever the management VM accesses them. Therefore, it is difficult to support para-virtualization because CloudVisor does not allow the management VM to access pages necessary for VM management in user VMs, e.g., the P2M table in Xen. The security monitor cannot recognize such pages by hardware events such as VM exit. Although CloudVisor supports VM migration for full virtualization, there are no performance data. In addition, CloudVisor introduces extra overheads to user VMs due to an additional virtualization layer. It does not trust even the VMM, but we believe that we can also trust the VMM if we can trust the security monitor via remote attestation.

The secure runtime environment (SRE) [3], [8] preserves the confidentiality of user VMs for para-virtualized operating systems. Like VMCrypt, SRE encrypts memory pages except several pages necessary for VM management when the management VM accesses them. SRE supports boot, suspend, and resume of user VMs. However, it cannot enable live migration because it provides an encrypted view to the management VM only when a user VM is not running. Moreover, SRE needs to modify the existing management software so that the VMM can identify all the unencrypted pages. For example, the resume program has to notify the VMM of the pages used for the P2M table because the VMM cannot identify them at the resume time.

In VMware vSphere Hypervisor (ESXi) [9], the VMM itself includes the management functions and the system administrators perform VM management by sending commands directly to the VMM. Unlike VMware ESX, the management VM called the service console is not used by default. Lacking the service console makes it difficult to steal information from the user VMs' memory. However, this architecture lowers the flexibility of the VM management. It is not easy for the administrators to use their own custom management software.

Domain disaggregation [11] moves management functions in the management VM to another VM called DomB. This architecture reduces the privileges of the management VM and prevents the attackers in the management VM from stealing sensitive information in the user VMs' memory. For example, most of the code for building VMs is run in DomB. However, the administrators have to be still trusted because they also manage DomB as well as the management VM. In addition, domain disaggregation needs a large modification to the existing management software so as to use the functions provided by DomB.

Overshadow [12] provides a dual memory view using the VMM. The basic idea is similar to VMCrypt, but Overshadow provides a normal view for applications and an encrypted view for the guest operating system. The aim of Overshadow is to prevent information leakage from applications to the operating system. Furthermore, SP^3 [13] can control views among applications as well. In contrast, VMCrypt prevents information leakage from the user VMs to the management VM. Particularly, the mechanisms for enabling live migration are specific to VMs.

3. VMCrypt

VMCrypt prevents the sensitive information of the user VMs' memory from leaking via the management VM, using the trusted VMM. Nevertheless, the IaaS administrators can manage the user VMs with the management VM, including live migration, as before VMCrypt is introduced. VMCrypt supports para-virtualized guest operating systems and allows the IaaS administrators to use the existing management software.

3.1 Threat Model and Assumptions

We assume that the management VM can be compromised by outside attackers or abused by IaaS administrators. Such attackers could take the root privilege of the operating system in the management VM and even alter the operating system kernel. This means that the management VM is removed from the trusted computing base in terms of confidentiality.

In this paper, we focus on the attempts to steal sensitive information from the user VMs' memory. Information leakage from other resources such as CPU registers, storage, networks, and covert channels is out of scope. In real IaaS clouds, these resources should be protected against the management VM as well. Since there are several studies for protecting them, we can incorporate those with VMCrypt. For example, the protection of CPU registers has been proposed in SRE [3], [8]. Storage and networks can be encrypted by the guest operating systems themselves, as described in Section 2, or by the VMM [14]. Mitigating the risk of covert channels is discussed in Ref. [15]. Also, we do not consider the other types of attacks against the user VMs from the management VM, such as integrity attacks and denial-of-service attacks.

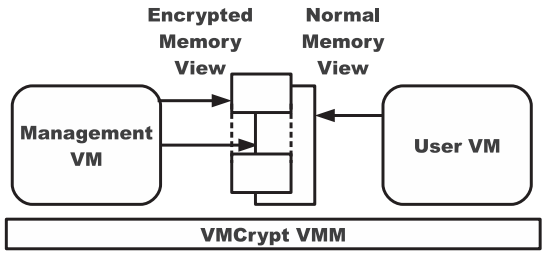


Fig. 1 A dual memory view provided by VMCrypt.

We assume that IaaS providers themselves are trusted, as widely accepted [2], [4]. In IaaS, trusted senior administrators should be responsible for the maintenance of VMMs and the hardware. They would not be lazy or malicious, unlike average IaaS administrators that manage the user VMs in the management VM. In real IaaS clouds, average administrators may manage VMMs and the hardware as well, but senior administrators should finally examine the correctness. Therefore, we assume that VMMs are well maintained and have no vulnerabilities that are compromised by the attackers. Also, we do not consider physical attacks such as the cold boot attack [16] because server rooms should be strictly protected in data centers.

3.2 Dual Memory View

VMCrypt provides a dual memory view for each user VM: a normal view and an encrypted view, as illustrated in **Fig. 1**. A *normal* view is a view of unencrypted memory and is used by a user VM. This view enables software running inside a user VM to access its memory as usual. In contrast, an *encrypted* view is a view of encrypted memory and provided to the management VM. Management software running in the management VM can access only encrypted data via this view, so that the attackers in the management VM cannot steal any useful information inside the user VMs. Nevertheless, the IaaS administrators in the management VM can manage the user VMs through the encrypted views. For example, migrating a user VM is achieved by transferring the encrypted view of the VM's memory to the destination.

To enable the IaaS administrators to manage the user VMs with para-virtualization, VMCrypt exceptionally provides a normal view to the management VM only for several memory regions. For such regions, the management VM can directly access unencrypted data in a user VM. Examples of such regions are memory shared between VMs and the page tables inside a user VM. For para-virtualized guest operating systems, the management VM needs to read and write data in the shared memory to exchange information with a user VM. The page tables have to be modified by the management VM during VM migration. The details of such unencrypted memory regions are listed in Section 4.4.

VMCrypt automatically identifies such unencrypted memory regions by monitoring the interaction between the management VM, the user VMs, and the VMM. For example, a shared memory region is passed from the management VM to the user VM

via the VMM. The memory regions for the page tables are registered to the VMM. As such, the VMM can recognize all the unencrypted memory regions. The information on the identified memory regions is cached during the life cycle of a user VM, including VM migration, to preserve the compatibility with the existing management software. With the cache, the VMM can restore the user VM's memory correctly even at another host.

The two views in a dual memory view are provided *concurrently* to enable live migration. In other words, an encrypted view coexists with a normal one for each user VM. Therefore, the management VM can access the memory of a running user VM. Live migration requires that the management VM transfers the memory of a user VM while the user VM accesses its memory. If VMCrypt directly encrypted the memory of the user VM by overwriting it, the running software in the user VM would fail when reading encrypted data.

3.3 Trusted VMMs and Coordinator

To allow only the trusted VMM to be run, VMCrypt performs remote attestation with a trusted server called the *trusted coordinator* (*TC*). The TC runs in a trusted third party outside IaaS clouds. Remote attestation certifies the authenticity of the VMM by tamper-resistant hardware such as the trusted platform module (TPM) [17]. It measures the VMM by calculating its hash value, sends the signed measurement to the TC, and verifies its integrity. According to our threat model, trusted senior administrators in IaaS configure the VMM correctly for remote attestation. Average IaaS administrators that may not be trusted cannot bypass this process.

The trusted VMM maintains two types of cryptographic keys: session keys and a pair of public and private keys. A session key is maintained for each user VM. It is used to construct an encrypted view from a normal one of user VM's memory and vice versa. A pair of public and private keys is used to encrypt and decrypt the session keys. The public key is securely registered to the TC. Since the session keys and the private key are kept inside the trusted VMM, they are not stolen by the attackers in the management VM. Also, they are not extracted from the binary file of the VMM stored in a disk. The session keys are dynamically generated at runtime. The private key is sealed by the TPM and only the securely-booted VMM can unseal it.

A session key for a user VM has the following life cycle. The VMM generates a new session key whenever a user VM is created whereas it discards the key at the destruction time of the VM. When a user VM is migrated, the VMM transfers the encrypted session key from a source host to a destination host, as proposed in TCCP [2]. **Figure 2** shows the procedure. First, host A's VMM obtains the public key of host B's VMM securely from the TC. Since the public key is signed with TC's private key, host A's VMM can verify it. The TC's public key is embedded into the VMM in advance to prevent man-in-the-middle attacks. Next, host A's VMM encrypts the session key of the target VM with B's public key and transfers it to host B. Host B's VMM decrypts it using B's private key and shares the session key for the migrating VM with host A's VMM. As such, session keys are passed to only hosts authorized by the TC.

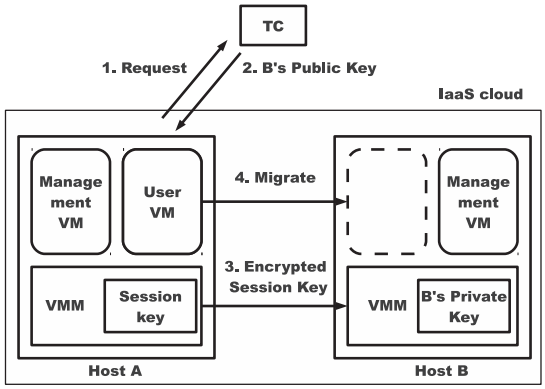


Fig. 2 The management of session keys on VM migration.

4. Implementation

We have implemented VMCrypt in Xen 4.0.1 [5]. In Xen, a user VM is domain U and the management VM is domain 0. In the current implementation, VMCrypt supports para-virtualized Linux for the x86-64 architecture as guest operating systems. VMCrypt mainly depends on Xen in how the VMM identifies unencrypted memory regions. If we implement this method in the VMM, VMCrypt can be applied to full-virtualization in Xen and the other virtualized systems such as VMware ESX.

4.1 Memory Model in Xen

Xen distinguishes machine memory and pseudo-physical memory to virtualize memory. *Machine memory* is the entire memory installed in the machine and consists of a set of machine page frames. It is reserved for the VMM, allocated to domains, or unallocated. Each machine page frame has a number called the *machine frame number* (*MFN*), which is consecutively numbered from 0. *Pseudo-physical memory* is a per-domain abstraction and allows a guest operating system to consider the allocated physical pages as contiguous. For each machine page frame, a *pseudo-physical frame number* (*PFN*) is consecutively numbered from 0. The VMM maintains a machine-to-physical (M2P) table for the mapping from MFNs to PFNs. For the inverse mapping, a physical-to-machine (P2M) table is maintained by each domain.

4.2 Constructing an Encrypted View

To construct an encrypted view from a normal view of domain U's memory, the VMM encrypts the contents of memory pages of domain U when domain 0 maps those pages. Domain 0 has to map memory pages on its address space to access domain U's memory. To make two memory views coexist, the VMM replicates pages of domain U and maps them on domain 0, as illustrated in **Fig. 3** (a). For this *encrypted replication*, the VMM allocates pages in domain 0 and copies encrypted contents to them. When domain 0 unmaps domain U's pages, the VMM decrypts the contents, writes them back to the original pages in domain U, and releases the pages allocated in domain 0.

The VMM detects memory mapping and unmapping of domain U by monitoring the modification to the page tables of domain 0. The page tables are maintained by the VMM and protected to prevent illegal memory accesses. To modify a page ta-

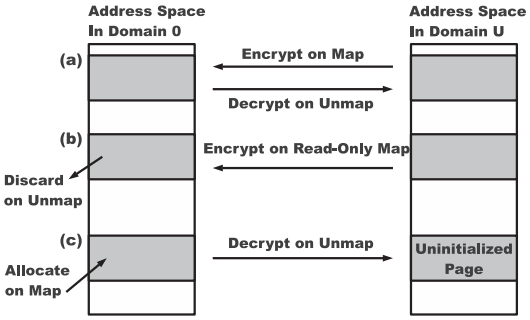


Fig. 3 Synchronization between an encrypted view and a normal view.

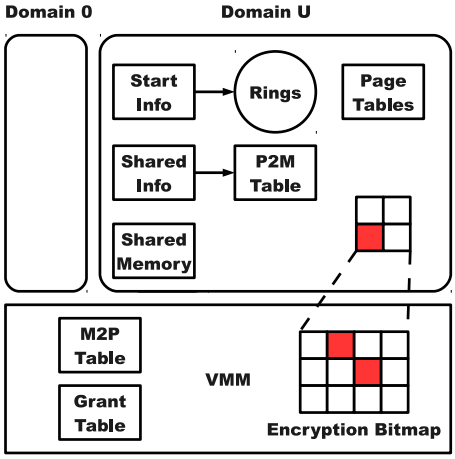


Fig. 4 Unencrypted pages and the encryption bitmap.

ble entry (PTE), domain 0 has to issue a hypercall to the VMM. If domain 0 attempts to modify its PTE directly, a page fault occurs and the VMM emulates the modification. In either case, the VMM can check the modification to PTEs. When a new PTE includes an MFN allocated to domain U, the VMM can notice that the modification is for memory mapping. When an old PTE includes an MFN belonging to domain U, the modification is for memory unmapping.

As an optimization for memory decryption, the VMM does not decrypt replicated pages when domain U's pages are mapped on domain 0 in a read-only manner (Fig. 3 (b)). Since domain 0 cannot modify read-only pages, the contents do not need to be written back to the original pages in domain U. When domain 0 unmaps such pages, the VMM simply releases them. This optimization is enabled by a dual memory view that concurrently exists because unencrypted contents are still preserved in the original pages. This reduces the overhead of memory decryption. For example, all of the domain U's pages are just read on suspending domain U.

As an optimization for memory encryption, the VMM does not encrypt domain U's uninitialized pages when those pages are mapped on domain 0 (Fig. 3 (c)). We define as *uninitialized* a page in which any data has not been written since domain U is created. Such uninitialized pages do not need to be encrypted because they contain no valid data. This reduces the overhead of memory encryption. For example, most of the domain U's pages are initialized by domain 0 when domain U is booted and resumed.

In this implementation, normal and encrypted views are synchronized only on memory mapping and unmapping by domain 0 for efficiency. We call this *lazy synchronization*. If domain U is stopped, the encrypted view is the latest because domain U does not modify the normal view. However, the encrypted view may become obsolete if domain U is running. For example, live migration is performed without stopping domain U. This inconsistency between memory views is usually acceptable because domain 0 cannot access domain U's memory consistently even when the memory pages are shared between domain 0 and domain U, as traditionally performed. Management software in domain 0 should already consider this.

4.3 Dealing with Unencrypted Pages

The VMM does not encrypt the contents of unencrypted pages, which domain 0 can access for the VM management. When domain 0 maps such pages, the VMM makes domain 0 share the pages with domain U. Domain 0 can read information from the shared pages at any time, but it can update the pages only before domain U starts running. This prevents domain 0 from interfering with running domain U, e.g., by altering the page tables. This limitation does not disable most of VM management because domain 0 only sets up the shared pages at the creation time of domain U. When domain 0 unmaps the pages, the VMM simply ceases to share them. This mechanism is the same as the traditional one for memory mapping between domain 0 and domain U.

To cache unencrypted pages that have been identified once, the VMM maintains the *encryption bitmap*, as shown in **Fig. 4**. Each bit corresponds to each machine page frame consecutively and all the bits are set at first. The bit is cleared if the corresponding page is identified as unencrypted. For the attempt to map domain U's pages, the VMM determines which view it provides to domain 0 by referring to this bitmap. The encryption bitmap is necessary because the VMM cannot always determine whether a specified page should be encrypted or not when the page is mapped on domain 0. The VMM identifies unencrypted pages at the appropriate times as described in Section 4.4 and constructs the encryption bitmap.

The encryption bitmap in the VMM is also embedded into domain U's memory. When domain U is migrated, the *embedded bitmap* is automatically transferred to the destination as well as the domain U's memory itself. This allows domain 0 to use the existing management software as is. Through the embedded bitmap, the VMM at the destination can extract information on unencrypted pages for the migrated domain U. Without the encryption bitmap, the VMM could not identify unencrypted pages when domain U is restored.

To allocate the bitmap inside domain U's memory, the VMM reserves a memory region using the e820 facility of BIOS. Since e820 is used to report the memory map to guest operating systems, the VMM inserts a reserved area into that memory map when it creates domain U. The encryption bitmap is copied to the reserved area in domain U's memory when those pages are mapped to domain 0. To preserve the integrity, the embedded bitmap is copied back on unmapping only while domain U is constructing, e.g., for VM migration. Its details are described

in Section 4.5.

4.4 Identifying Unencrypted Pages

The VMM automatically identifies the unencrypted pages to exceptionally deal with them. Figure 4 shows the data that domain 0 needs to access. This identification is specific to the para-virtualization in Xen.

4.4.1 Start Info

The start info page is used for passing information from domain 0 to domain U when domain U starts running. This page contains information necessary for booting a guest operating system such as the allocated memory size. Domain 0 needs to set up this page when domain U is booted or restored.

The VMM identifies the start info page by monitoring virtual CPU registers of domain U. This identification is done when domain 0 issues the first unpause hypercall for the domain U. This hypercall is used to start running domain U. To pass the start info page to domain U, domain 0 sets the virtual address of the start info page in domain U to the RSI register at boot time. The VMM translates the virtual address to the corresponding MFN. Even if the attackers alter that register, they cannot steal useful information. They could make the VMM recognize an arbitrary page as the start info page. At this time, however, domain U will not be able to boot because an arbitrary page does not contain the correct information for booting. In addition, domain U has no sensitive information yet at the boot time.

4.4.2 Console/XenStore Rings

One pair of ring buffers is used to achieve the console of domain U. Domain 0 needs to read text outputs from and write key inputs to the console. The other pair is used for domain U to access a storage system called XenStore in domain 0. Domain 0 and domain U exchange information on VM configurations through XenStore.

Since the information on these ring buffers is stored in the start info page, the VMM can identify it easily. Even if the attackers specify arbitrary pages as the ring buffers, domain U uses them only for console and XenStore, not for the other purposes. Domain U may write sensitive information to the console, and so the contents should be encrypted between domain U and its user. Such a mechanism for secure console is beyond the scope of this paper.

4.4.3 Shared Info

The shared info page is used for sharing information among the VMM, domain U, and domain 0. Through this page, the VMM notifies domain U of virtual CPU interrupts, and so on. Domain U passes the P2M table to domain 0. The VMM can easily identify the page because the VMM itself allocates the page when it creates domain U.

4.4.4 P2M Table

The P2M table is a mapping table from PFNs to MFNs. The P2M table has a tree structure to allow machine page frames to be sparsely allocated to domain U. Domain 0 needs to access this table to obtain all the MFNs allocated to domain U when it migrates domain U. The VMM identifies the top page of the P2M table from the shared info page. It traverses the P2M table from the top node and recognizes all the pages used for the table. To prevent the attackers from making arbitrary pages be a part of the P2M table, the VMM checks the data structure and the validity of all the entries at this time. Note that the VMM has to perform this traversal whenever domain 0 maps the shared info page of domain U. This is because the P2M table is constructed by the guest operating system in domain U.

4.4.5 Page Tables

In para-virtualized operating systems, a page table is a table for translating virtual addresses into MFNs, not PFNs. When domain 0 migrates domain U, it needs to rewrite all the PTEs in the page tables of the domain U, using the MFNs at the destination host. The VMM identifies the pages used for the page tables in domain U through a hypercall. The pages are always registered to the VMM. If a page is no longer used for a page table, it is unregistered from the VMM. The attackers cannot register arbitrary pages as page tables because the VMM checks the validity of all the PTEs. In addition, the VMM disallows domain 0 to modify the page tables after domain U starts running.

4.4.6 Shared Memory with the Grant Table

The grant table is a mechanism for sharing memory pages between domain U and domain 0. For example, I/O ring buffers are shared between the frontend drivers in domain U and the backend drivers in domain 0. With the grant table, domain 0 has to read and write the memory pages permitted by domain U. The VMM can identify all the shared pages by checking the grant table. Domain 0 can map only the pages that domain U explicitly permits to access. Such pages may include sensitive information, but domain U should encrypt it with encrypted file systems and VPN, as described in Section 2.

4.5 Live Migration with VMCrypt

In live migration, domain 0 transfers the memory image of running domain U from a source host to a destination host. VMCrypt encrypts the memory contents while it allows domain 0 to access necessary information in domain U's memory.

4.5.1 Source Host

Domain 0 first maps the shared info page of domain U to obtain information on the VM and the P2M table. Then it transfers the P2M table to the destination host. At this time, it canonicalizes the entries so that the table does not depend on host-specific memory allocation. Specifically, it replaces MFNs in the entries with the corresponding PFNs. Next, domain 0 maps all the pages of domain U and transfers their contents to the destination in turn. When domain 0 transfers pages used for page tables, it canonicalizes the PTEs in the tables. In live migration, domain 0 repeatedly transfers dirty pages, which are modified by domain U during migration. Finally, it stops domain U and transfers the remaining dirty pages and other states.

VMCrypt allows domain 0 to inspect domain U's memory pages necessary for migration, such as the shared info page, the P2M table, and the page tables. Such shared pages are not encrypted while the other pages are encrypted by the VMM when domain 0 maps them. Thanks to a dual memory view provided by VMCrypt, domain 0 can concurrently access encrypted pages while domain U accesses the original pages. When domain 0 unmaps pages, the VMM does not decrypt them for performance

because any pages are not modified by domain 0.

When domain 0 maps the pages used for the embedded bitmap, the VMM copies the encryption bitmap in the VMM to the pages and encrypts them. If the encryption bitmap changes during live migration, the embedded bitmap has to be re-transferred. The VMM makes the pages for the embedded bitmap dirty so that the migration software in domain 0 re-transfers them automatically. To detect the corruption of the bitmap during migration, the VMM embeds the hash value of the bitmap into the domain U's memory. The bitmap is critical because domain 0 can map random pages as unencrypted by corrupting the bitmap.

Also, the VMM embeds the hash value of the page tables into domain U's memory to detect the alteration during migration. For example, if restored domain U uses altered page tables, the domain U might store sensitive information to unencrypted pages. When domain U is finally stopped at the source host, the VMM canonicalizes all the entries and calculates its hash value.

4.5.2 Destination Host

At the destination host, domain 0 creates a new domain U and reconstructs its memory using the received memory image. When it receives a memory page, it allocates a new page for domain U, maps it, and writes the contents to it. When domain 0 receives pages used for page tables, it uncanonicalizes the PTEs of the tables, according to the memory allocation at the destination host. Specifically, it replaces PFNs in the PTEs with the corresponding MFNs. Domain 0 repeats this as long as memory pages are transferred from the source host. When domain 0 has received all data from the source, it sets up the start info and shared info pages. Then it uncanonicalizes the P2M table and finally starts domain U.

The VMM does not decrypt the received memory pages on memory unmapping, but just before domain U starts running. Until the encryption bitmap is restored, the VMM cannot determine whether each page should be decrypted or not. This delay of decryption does not cause any problems. During live migration, domain U at the destination host neither runs nor accesses any pages. Domain 0 can access shared pages such as the page tables because those pages are transferred without encryption from the source host. When domain 0 issues the first unpause hypercall to start domain U, the VMM extracts the embedded bitmap from domain U's memory. Then it copies back the bitmap to the encryption bitmap in the VMM. After the bitmap extraction, the VMM decrypts memory pages of domain U on the basis of the encryption bitmap.

As an optimization, the VMM decrypts memory pages as early as possible to reduce the downtime of live migration. If all pages are decrypted at the final stage, the downtime becomes long because domain U at the source host is stopped at this stage. The VMM extracts the embedded bitmap just after domain 0 receives all the pages used for the bitmap. After that, the VMM can decrypt memory pages on the basis of the restored encrypted bitmap. However, the encryption bitmap may not be consistent because it can be updated during live migration. For example, pages to be encrypted may be mapped without encryption before the encryption bitmap is correctly updated. This may lead to information leakage from domain U.

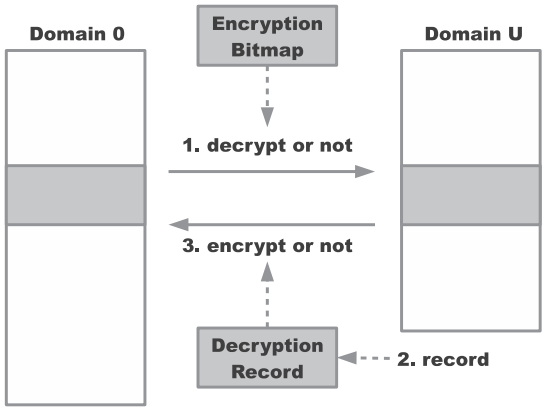


Fig. 5 Encryption based on the decryption record.

To prevent this inconsistency, the VMM maintains the *decryption record*, which is used to record whether each page of domain U has been decrypted or not, as shown in **Fig. 5**. Its bit is set when the corresponding page is decrypted. When domain 0 unmaps a page of domain U, the VMM sets the corresponding bit of the decryption record if the VMM decrypts it on the basis of the encryption bitmap. When domain 0 maps the page later, the VMM determines whether the page should be encrypted or not, using the decryption record instead of the encryption bitmap. With the decryption record, the VMM can give a consistent memory view to domain 0. It is guaranteed that decrypted pages are necessarily encrypted when domain 0 maps them. At the final stage, the VMM adjusts the encryption of all the pages on the basis of the consistent encryption bitmap. Note that unencrypted pages may be incorrectly decrypted but they can be restored by encryption in case of AES-XTS at least.

The attacks against the embedded bitmap cannot succeed in information leakage. Corruption of the bitmap is detected by the hash value embedded into domain U's memory. Replay attacks are useless because domain 0 is always given the same memory view as that at the source host. Even if a page is decrypted using an old encryption bitmap, it is necessarily encrypted again on the basis of the same bitmap when domain 0 maps it.

Alteration of the page tables is also detected by the embedded hash value. Just before domain U starts running, the VMM canonicalizes the PTEs, calculates the hash value, and compares it with the embedded one. Replay attacks are not impossible, but it is difficult to steal useful information with only old page tables. In addition, the pages used for replayed page tables have to be marked as unencrypted in the encryption bitmap, which cannot be compromised.

4.6 Other VM Management with VMCrypt

For boot, domain 0 can construct domain U as usual because the VMM does not encrypt any memory pages. It loads the kernel to domain U's memory and sets up the initial page table, the start info page, and so on. When domain U starts running, the VMM checks the integrity of domain U's kernel and the other data structure, as described in previous work [3], [8]. In addition, VMCrypt supports the suspension and resumption of domain U, which are similar to operations at the source and destination hosts in VM migration, respectively.

4.7 Security Consideration

In VMCrypt, the management VM could interfere with the VMM through the alteration of unencrypted pages of the user VMs. For example, when the VMM traverses the P2M table, it might crash if the table was altered by the attackers so that it includes non-existing memory pages. To prevent this, our VMM carefully checks that the accessed pages belong to the target VM.

VMCrypt increases the size of the VMM, resulting in a larger TCB. This may make the whole system more vulnerable, but the increased code size is 12,500 lines, including 6,500 lines of code for AES. This size is only 5% of the original VMM.

5. Experiments

We performed experiments to measure the overheads of VMCrypt and confirm that VMCrypt prevents information leakage via domain 0. We used two PCs, each of which has one Intel Xeon processor 2.67 GHz with 8 cores, 12 GB of memory, a 1 TB of SATA HDD, and a Gigabit Ethernet NIC. We ran modified Xen 4.0.1 for the x86-64 architecture on these PCs. For domain 0, we allocated 6 GB of memory and ran Linux 2.6.32-5-xen-amd64. For domain U, we allocated 1 GB of memory if not specified in each experiment and ran para-virtualized Linux 2.6.32.27. These PCs were connected with a Gigabit Ethernet switch.

Through the experiments, VMCrypt used AES-XTS with a key size of 256 bits for the encryption of domain U's memory. We have implemented the AES-XTS support in the VMM but not yet fully optimized. The performance can be improved by using AES-NI, a set of special instructions for AES. To exclude the overhead of cryptographic operations, we also used the null cipher, which did not encrypt or decrypt data. For comparison, we conducted the experiments in the vanilla Xen.

5.1 Overhead of Constructing an Encrypted View

To examine the overhead of constructing an encrypted view, we measured the time needed for mapping and unmapping a memory page of domain U on domain 0. We performed this experiment for writable mapping, read-only mapping, and the mapping of uninitialized memory. VMCrypt encrypts and decrypts a writable page, only encrypts a read-only page, and only decrypts an uninitialized page, respectively. We repeated memory mapping and unmapping 100,000 times. **Figure 6** shows the mean time.

When VMCrypt used the null cipher, the execution time increased by 3 μs in comparison with the vanilla Xen. This overhead comes from examining the encryption bitmap and replicating a page. The optimization for encrypted replication was not effective because the memory copies were completed only on the CPU cache. When VMCrypt performed both encryption and decryption of AES, the execution time was 81 μs, which was 7.7 times longer than that in the vanilla Xen. When the optimization was enabled, the execution time was reduced to less than 60% of the non-optimized case. This shows that our optimization for reducing cryptographic operations is effective.

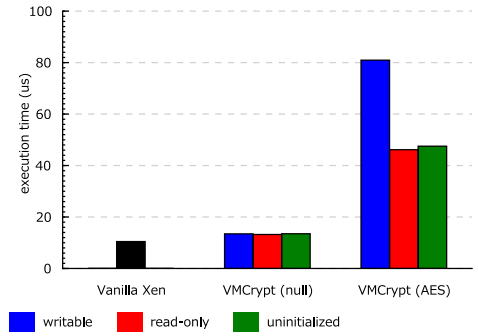


Fig. 6 Time used by the domain 0 to map a page of a domain U.

5.2 Memory Overhead for an Encrypted View

We examined the number of extra pages allocated for an encrypted view. When domain 0 maps domain U's pages, the VMM allocates new pages in domain 0 for encrypted replication. At worst, the number of the allocated pages could be equal to the total number of domain U's pages. However, the maximum number of extra pages was 1,024 while we were migrating domain U. This is because the migration program mapped 1,024 pages at once and unmapped all of them after the memory manipulation. 1,024 pages are 4 MB of memory and not so large.

5.3 Performance of Live Migration

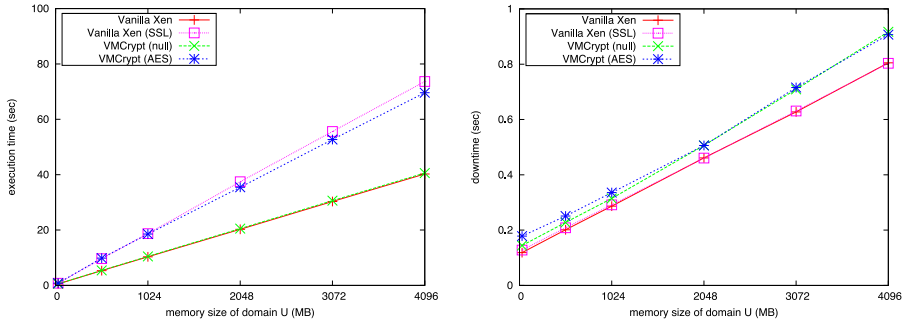
With VMCrypt, we performed live migration, which transfers the memory image without stopping domain U and reduces the downtime of domain U. For comparison, we also performed migration with SSL. Xen provides the mechanism of migrating domain U via an SSL connection. Although SSL can prevent information leakage on the network, domain 0 at both hosts can still steal information. Transferred data on the SSL connection is decrypted at domain 0. Xen used AES with a key size of 256 bits for encryption.

We measured the time needed for migrating domain U between two hosts. We changed the allocated memory size of domain U from 26 MB to 4 GB. The results are shown in **Fig. 7** (a). The migration time was proportional to the memory size of migrated domain U. The overhead of VMCrypt with the null cipher was only 1%. With AES, however, the migration time was 1.7 times longer than that in the vanilla Xen. These results mean that encryption and decryption degraded the performance largely. Note that the performance in VMCrypt with AES is almost the same as that in the vanilla Xen with SSL. The reason why the vanilla Xen with SSL is slightly slower than VMCrypt with AES is that SSL checks the data integrity as well.

Next, we measured the downtimes due to live migration of domain U. Here, the downtime is the time from when domain U is stopped at the source host until it is restarted at the destination host. As shown in Fig. 7 (b), the downtime was also proportional to the memory size of domain U. In VMCrypt with AES, it was still less than one second even when the memory size was 4 GB. It was 13% longer than that in the vanilla Xen.

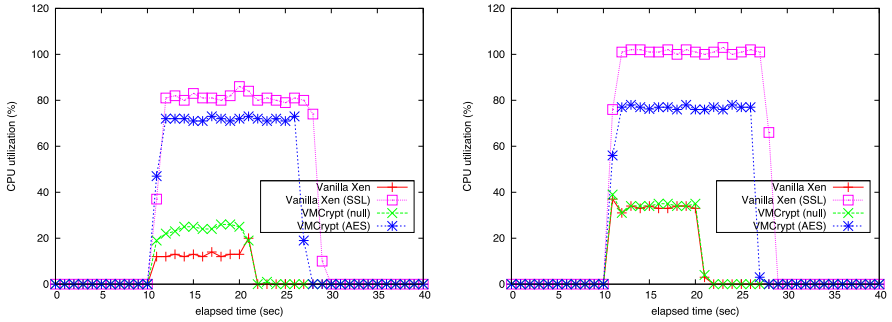
Third, we measured the number of re-transferred pages, which increases due to the re-transfer of the embedded bitmap. On average, 29.5 extra pages were re-transferred, but this is negligible when compared to the total number of pages of domain U.

Finally, we measured the CPU utilization of domain 0 during live migration. **Figure 8** (a) and (b) show the CPU utilization of domain 0 at the source and destination hosts, respectively.



(a) Migration time (b) Downtime

Fig. 7 The performance of live migration.



(a) Source host (b) Destination host

Fig. 8 The CPU utilization during live migration.

The measured CPU time includes the CPU time consumed in the VMM by hypercalls issued by domain 0. Compared with the vanilla Xen, the CPU utilization in VMCrypt with AES only increased by 35% and 30% at the source and destination hosts, respectively. However, that in the vanilla Xen with SSL increased by 70% and 65%, respectively. One of the reasons of such high CPU utilization with SSL is the integrity check.

5.4 Performance of Other VM Management

First, we measured the time needed for building domain U. The time we measured was from when we started creating domain U until the VMM completed the unpause hypercall. The overhead due to VMCrypt was only 1% when 4 GB of memory was allocated. This is because the VMM does not encrypt or decrypt at boot time.

Next, we measured the times for suspension, resumption, and migration, using VMCrypt with AES. The resumption and migration times were 1.8 and 2.3 times longer than those in the vanilla Xen, respectively. These overheads come from cryptographic operations of AES. Interestingly, the overhead for suspension was only 7% because the overhead of AES was hidden by disk I/O.

5.5 Performance Degradation of Domain U

We measured the impact of VMCrypt on the performance of domain U. VMCrypt does not perform encryption or decryption unless domain 0 maps or unmaps pages of domain U, but the VMM checks several operations even when domain U modifies page tables. We ran two benchmarks, lmbench [18] and UnixBench [19], on domain U in the vanilla Xen and VMCrypt. **Figure 9** shows the performance degradation in lmbench due to VMCrypt. Context switching in VMCrypt was 1.8% slower than

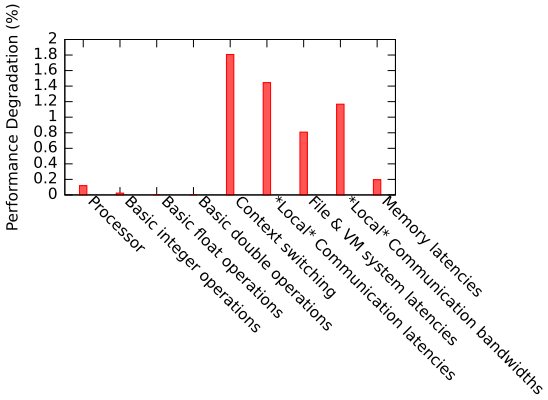


Fig. 9 Performance degradation of domain U by VMCrypt.

in the vanilla Xen, but the overall average of performance degradation in VMCrypt was 0.7%. For UnixBench, the overall average of performance degradation in VMCrypt was also 0.7%.

Next, we examined how live migration in VMCrypt affected the performance of a web server running in domain U. We ran the lighttpd web server [20] in domain U and measured its throughput with ApacheBench [21] in a client host. The client host had one Intel Core 2 Quad processor 2.83 GHz, 8 GB of memory, and a Gigabit Ethernet NIC. The server and client hosts were connected with a Gigabit Ethernet switch. **Figure 10** shows that the throughput in VMCrypt with AES was higher than that in the vanilla Xen. This is because live migration in the vanilla Xen occupies the network bandwidth and the web server cannot use the network sufficiently. However, the throughput in VMCrypt with AES is lower than that in the vanilla Xen with SSL. In VMCrypt, the sender transfers 4 MB of the memory image at once and causes bursty network traffic intermittently. This bursty traf-

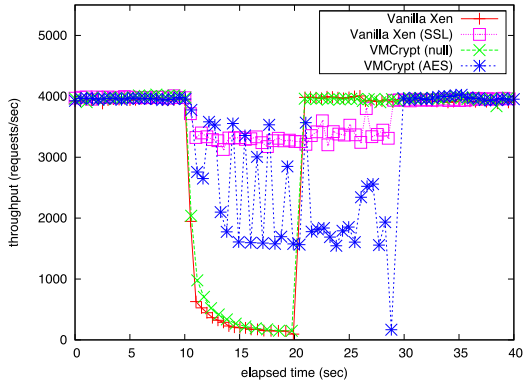


Fig. 10 The throughput of a web server in domain U during live migration.

fic affects the throughput of the web server. With SSL, in contrast, the traffic is not bursty because the sender slowly transfers the data while encrypting.

5.6 Overhead of Remote Attestation

We examined the overhead of remote attestation, which is performed only when a host is booted. First, we measured the time needed for the measurement of the VMM using TrustedGRUB 1.1.5 [22]. TrustedGRUB is a boot loader that calculates the SHA-1 hash value of not only the VMM but also the kernel in domain 0 and stores the value in the TPM. We compared the reboot time for TrustedGrub with that for GRUB Legacy. The time for the measurement was 0.5 sec.

Next, we attempted the verification of the measurement by the TC, but OpenPTS [23] did not work in our experimental environment. According to the literature [24], the verification process takes about one second. The time should depend on the network performance.

5.7 Leakage Tests with VMCrypt

We confirmed that VMCrypt prevented information leakage from domain U's memory. First, we attempted an attack to find AES shared keys used by OpenSSH processes. We logged in domain U to make the SSH server generate an AES key, obtained the memory dump of the domain U, and used the `aeskeyfind` tool [16] to find AES keys from the memory dump. Second, we attempted an attack to find RSA private keys generated by OpenSSL. We generated an X.509 certificate signing request by executing the `openssl` command in domain U and used the `rsakeyfind` tool [16]. We could find several keys without VMCrypt, but VMCrypt could successfully prevent such tools from finding keys.

For the page cache, we attempted an attack to obtain shadow passwords from the page cache in domain U's memory. We logged in domain U as root, so that the contents of `/etc/shadow` were stored in the page cache. Then we searched the memory dump of the domain U for the string "`root:$`," with which a root password begins. VMCrypt disabled such a password search whereas we could find a root password when the page cache was not encrypted by VMCrypt.

6. Conclusion

In this paper, we proposed VMCrypt, which preserves the data secrecy of user VMs' memory in IaaS clouds. VMCrypt provides a dual memory view concurrently and prevents sensitive information from leaking to the management VM. It also allows the administrators to use the existing management software including live migration. To support the existing management software for para-virtualized operating systems, VMCrypt automatically identifies unencrypted memory regions and caches that information during the life cycle of user VMs. We have implemented VMCrypt in the Xen VMM and confirmed that the administrators could manage user VMs as usual but could not steal the sensitive information. In particular, the downtime due to live migration was comparable with the vanilla Xen and still less than one second.

One of the future work is to reduce the overhead of memory decryption. In the current implementation, pages mapped on the management VM are synchronously decrypted on unmapping. To overlap the decryption with I/O, the VMM should decrypt unmapped pages asynchronously. Another future work is to enable VM introspection from the management VM. To introspect the user VMs, the management VM has to access user VMs' memory without encryption. We are planning to implement the support for VM introspection based on permission by the users. Supporting full-virtualization is also necessary for applying VMCrypt to real IaaS clouds. The VMM has to identify the memory regions used for framebuffers and DMA as unencrypted pages, which is done by CloudVisor [4].

References

[1] Amazon Web Services: Amazon Elastic Compute Cloud, available from ⟨http://aws.amazon.com/ec2/⟩.
[2] Santos, N., Gummadi, K.P. and Rodrigues, R.: Towards Trusted Cloud Computing, *Proc. Workshop on Hot Topics in Cloud Computing* (2009).
[3] Li, C., Raghunathan, A. and Jha, N.K.: Secure Virtual Machine Execution under an Untrusted Management OS, *Proc. Intl. Conf. Cloud Computing*, pp.172–179 (2010).
[4] Zhang, F., Chen, J., Chen, H. and Zang, B.: CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization, *Proc. Symp. Operating Systems Principles*, pp.203–216 (2011).
[5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *Proc. Symp. Operating Systems Principles*, pp.164–177 (2003).
[6] Clark, C., Franser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live Migration of Virtual Machines, *Proc. Symp. Networked Systems Design and Implementation*, pp.273–286 (2005).
[7] Nelson, M., Lim, B. and Hutechins, G.: Fast Transparent Migration for Virtual Machines, *Proc. USENIX Annual Technical Conf.*, pp.391–394 (2005).
[8] Li, C., Raghunathan, A. and Jha, N.K.: A Trusted Virtual Machine in an Untrusted Management Environment, *IEEE Trans. Services Computing*, IEEE Computer Society Digital Library (2011).
[9] VMware, Inc.: VMWare, available from ⟨http://www.vmware.com/⟩.
[10] Rocha, F. and Correia, M.: Lucy in the Sky without Diamonds: Stealing Confidential Data in the Cloud, *Proc. Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments*, pp.129–134 (2011).
[11] Murray, D.G., Milos, G. and Hand, S.: Improving Xen Security through Disaggregation, *Proc. Intl. Conf. Virtual Execution Environments*, pp.151–160 (2008).
[12] Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger,

C.A., Boneh, D., Dwoskin, J. and Ports, D.R.K.: Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems, *Proc. Intl. Conf. Architectural Support for Programming Languages and Operating Systems*, pp.2–13 (2008).
[13] Yang, J. and Shin, K.G.: Using Hypervisor to Provide Data Secrecy for User Applications on a Per-Page Basis, *Proc. Intl. Conf. Virtual Execution Environments*, pp.71–80 (2008).
[14] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y. and Kato, K.: BitVisor: A Thin Hypervisor for Enforcing I/O Device Security, *Proc. Intl. Conf. Virtual Execution Environments*, pp.121–130 (2009).
[15] Ristenpart, T., Tromer, E., Shacham, H. and Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds, *Proc. Conf. Computer and Communications Security*, pp.199–212 (2009).
[16] Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J. and Felten, E.W.: Lest We Remember: Cold Boot Attacks on Encryption Keys, *Proc. USENIX Security Symp.*, pp.45–60 (2008).
[17] Trusted Computing Group: TPM Main Specification Version 1.2, available from ⟨http://www.trustedcomputinggroup.org/⟩.
[18] McVoy, L. and Staelin, C.: lmbench, available from ⟨http://www.bitmover.com/lmbench/⟩.
[19] Smith, I.: UnixBench, available from ⟨http://code.google.com/p/byte-unixbench/⟩.
[20] J. Kneschke: lighttpd, available from ⟨http://www.lighttpd.net/⟩.
[21] The Apache Software Foundation: Apache HTTP Server Benchmarking Tool, available from ⟨http://httpd.apache.org/⟩.
[22] Sirrix AG: TrustedGRUB, available from ⟨https://projects.sirrix.com/trac/trustedgrub/⟩.
[23] Munetoh, S.: Open Platform Trust Services, available from ⟨http://sourceforge.jp/projects/openpts/⟩.
[24] Dixon, C., Uppal, H., Brajkovic, V., Brandon, D., Anderson, T. and Krishnamurthy, A.: ETTM: A Scalable Fault Tolerant Network Manager, *Proc. Symp. Networked Systems Design & Implementation*, pp.85–98 (2011).

Hidekazu Tadokoro received his B.Sc. and M.Sc. degrees from Tokyo Institute of Technology in 2007 and 2009, respectively. He is now a Ph.D. student at Tokyo Institute of Technology. His current research interests are operating systems, virtual machines, and system software.

Kenichi Kourai received his Ph.D. degree from the University of Tokyo in 2002. He entered NTT Corporation in the same year. He became a research associate at Tokyo Institute of Technology in 2003. He is an associate professor at Kyushu Institute of Technology since 2008. His current research interests include operating systems, security, and system software.

Shigeru Chiba received his Ph.D. degree from the University of Tokyo 1996. He became an assistant professor at University of Tsukuba in 1997 and at Tokyo Institute of Technology in 2001, and a professor at Tokyo Institute of Technology in 2008. He is a professor at the University of Tokyo since 2011. His research interests include system software and programming languages.