IEICE TRANS. INF. & SYST., VOL.E98–D, NO.9 SEPTEMBER 2015

LETTER

# Isolated VM Storage on Clouds*

**Jinho SEOL**[†a)], ***Student Member***, **Seongwook JIN**[†], ***and*** **Seungryoul MAENG**[†], ***Nonmembers***

**SUMMARY** Even though cloud users want to keep their data on clouds secure, it is not easy to protect the data because cloud administrators could be malicious and hypervisor could be compromised. To solve this problem, hardware-based memory isolation schemes have been proposed. However, the data in virtual storage are not protected by the memory isolation schemes, and thus, a guest OS should encrypt the data. In this paper, we address the problems of the previous schemes and propose a hardware-based storage isolation scheme. The proposed scheme enables to protect user data securely and to achieve performance improvement.

***key words:*** *cloud computing, storage device, storage isolation*

## 1. Introduction

With a cloud computing service, cloud users can use computing resources by leasing them rather than owning them. To manage computing resources efficiently, a cloud provider allocates virtual machines (VMs) to cloud users. Enabling several VMs to co-exist in a single cloud node, virtual machine monitor (VMM; hypervisor) provides management for the resources and isolation between VMs. However, in such a virtualized environment, it is hard to protect the private data of cloud users because VMM can be compromised due to its vulnerabilities and can misbehave with a root privilege of cloud administrators. To protect private data of cloud users against such an environment, several studies have been proposed [1]–[3]. As the prior studies protect the memory of guest VMs by hardware isolation, the memory of guest VMs can be protected even though VMM is compromised.

Even though such an approach protects the memory of guest VMs efficiently, the hardware does not support any mechanism for storage devices. Thus, guest VMs are in charge of protecting their private data by encrypting them. Even though the encryption scheme enables to protect the data against a compromised VMM and malicious cloud administrators, there exist two problems. First, even though user data protection should be provided by a cloud service provider, a guest OS has a duty of protecting the data. Second, overheads are incurred during cryptographic operations.

To deal with such problems, this paper proposes a hardware-based storage isolation architecture called H-SVM+, which extends the previously proposed memory isolation hardware [1], [3]. The proposed architecture isolates storage access from VMM and cloud administrators, and the private data of cloud user can be protected without cryptographic operations. With the proposed architecture, a cloud user can use VM storage transparently and be free from cryptographic overheads.

Manuscript received February 26, 2015.
Manuscript publicized June 8, 2015.

[†]The authors are with the Computer Science Department, KAIST, Korea.

[*]This work was supported by the IT R&D program of MSIP/IITP [10041313, UX-oriented Mobile SW Platform].

a) E-mail: jhseol@calab.kaist.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2015EDL8048

## 2. Motivation and Background

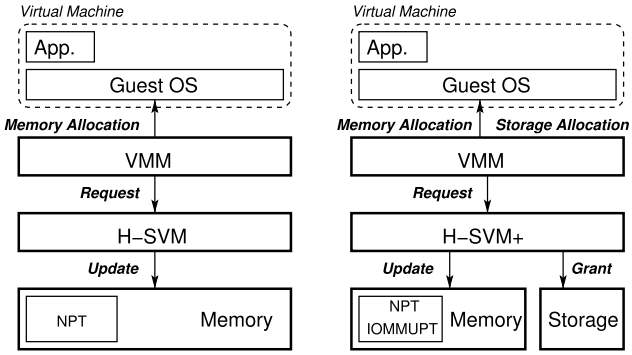
### 2.1 Assumption and Threat Model

VMM plays the essential role of VM isolation and resource allocation. The modern VMM has been constantly increasing its functionality, and thus, it becomes more complex. Such complex functionality makes VMM more powerful, while it makes VMM more vulnerable because it is hard to verify a huge body of code [4]. Therefore, our threat model assumes that VMM is untrustworthy and may try to leak private data of guest VMs. Moreover, an individual cloud administrator may be malicious whereas a cloud service provider tries to protect cloud user data.

Since this paper proposes a hardware-based architecture, the architecture cannot resist hardware attacks such as bus-probing and cold-boot attacks. However, a cloud provider equips proper defense mechanisms such as surveillance systems and access doors to prevent hardware attacks. Therefore, we assume that hardware attacks are unfeasible.

### 2.2 Hardware Based Memory Isolation

Since VMM can be compromised and cloud administrators can be malicious, hardware based memory isolation schemes such as H-SVM [1], [3] and HyperWall [2] have been proposed to provide strong memory isolation. A traditional VMM is in charge of both resource allocation and protection functions whereas such schemes decouple memory protection functions from VMM and move the functions into hardware as depicted in Fig. 1 (a). Accordingly, the memory of guest VMs can be protected even with a malicious VMM. However, since the storage of guest VMs is not directly protected by the proposed hardware, an OS in guest VMs has a duty for protecting the storage data with cryptographic operations.




(a) H–SVM (b) H–SVM+

**Fig. 1** Memory and storage isolation

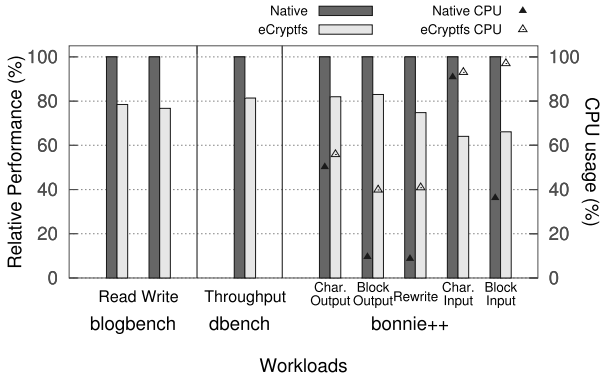



**Fig. 2** Performance of cryptographic operations

## 2.3 Implications of Cryptographic Operations

With a hardware based memory isolation architecture, guest VMs should perform cryptographic operations for storage data. The hardware supports special storage where cryptographic keys are stored and delivers the key to an OS of a guest VM. Then, the OS performs actual cryptographic operations with the key. Such an architecture has two implications for cloud users. First, the responsibility of protecting user data is unfairly placed to cloud users. Even though cloud users does not perform cryptographic operations for their data, the data should be protected. However, the previous schemes merely rely on the cryptographic operations by the guest OS. It means that a cloud user is responsible for protecting their storage data. Shifting the responsibility onto cloud users may result in an evasion of responsibility of a cloud provider. When information leakage happens on clouds, the cloud provider can insist that the root cause of the information leakage is the vulnerability of an guest OS or the neglect of cloud user's duty of managing the guest OS. Accordingly, a new architecture should be responsible for protecting both memory and storage data.

Second, the cost of protecting data is unfairly passed on to cloud users. Cryptographic operations should be executed whenever a guest VM reads or writes data. Accordingly, a cloud user suffers from a relatively low performance and should pay additional costs for cryptographic operations. To evaluate the overhead, we measured the performance of cryptographic operations. The experiment was performed on Intel i7-2600 with 1TB HDD, and eCryptfs [5] was used for an encryption scheme. Three different I/O benchmarks called blogbench [6], dbench [7], and bonnie++ [8] were used and the performance results were normalized to the performance of native system without cryptographic operations. Figure 2 shows performance overhead caused by cryptographic operations. The cryptographic operations lead to performance degradation up to 35%. Bonnie++ provides also CPU usage during its execution and the CPU usage is denoted as triangle marks in Fig. 2. The result shows that performance degradation and high CPU usage are inevitable due to cryptographic operations. Moreover, integrity check schemes aggravate such overheads. The overheads are equivalent to additional costs for cloud user to protect storage data. However, it is unfavorable for cloud users that they pays costs to protect their data even though cloud providers are responsible for protecting cloud data. Therefore, a new architecture should remove such overheads efficiently.

## 2.4 Requirements for Storage Isolation

To isolate storage data without cryptographic operations, virtual storage should be dedicated to a VM. To enable such a system, there exists three requirements. Firstly, a strong memory isolation scheme should be provided. Since the contents of storage devices are loaded in the memory of guest VMs, isolating the memory of guest VMs are essential. Secondly, once a storage device is allocated to a VM, the VM should have an exclusive access to the device. If an arbitrary VM is able to access to the device, the data in the storage device are disclosed. Thirdly, the communication path between a storage device and its device driver in a VM should be protected. Interrupts, register, and shared memory are used for the communication [9]. If a malicious entity is able to access the communication path, the confidentiality of the storage device cannot be guaranteed.

Since our system is built on a hardware-based memory isolation scheme, the first requirement is satisfied. To satisfy the third requirement, the storage devices which are directly assignable to a guest VM are used in the proposed system. One example of directly assignable devices is Non-Volatile Memory Express (NVMe) [10] with a Single Root I/O Virtualization (SR-IOV). NVMe is a host controller interface utilizing PCIe based sold state drives (SSD) and, SR-IOV is the PCIe specification for virtualization support. With SR-IOV capability, a physical device provides multiple virtual devices and each virtual device can communicate to a device driver without the intervention of VMM. In this paper, our architecture manages the mapping table to denote relationship between a VM and a storage device, and manages I/O Memory Management Unit (IOMMU) based on SR-IOV storage devices to protect the communication path.

## 3. Storage Isolation Architecture

H-SVM+ provides a storage isolation scheme. In this section, we focus on how the proposed architecture satisfies the second requirement in §2.4. To guarantee an exclusive access from a VM to virtual storage, H-SVM+ manages a table with identities (IDs) of VMs and storage devices to denote access permission as described in Fig. 3. Since H-SVM+ manages IOMMU page table (IOMMUPT), H-SVM+ is able to grant access permission from a VM to a storage device as far as H-SVM+ can check the IDs of VMs and storage devices. To identify VMs and storage devices, we confront three challenges.

The first challenge is how to determine an ID of a VM. The ID should be unique and cannot be changed arbitrarily, and thus, the VMID used by VMM is not suitable for identifying VMs. Therefore, in the proposed system, a VM is identified by the hash value of a VM. A virtual storage of guest VMs consists of two partitions. The first partition contains a guest OS and does not have user data. The second partition contains user data, and H-SVM+ focuses on the isolation of the second partition. Since H-SVM verifies and launches the first partition [1], H-SVM knows the hash value of the first partition. In the proposed system, the first partition includes a user name and nonce to differentiate the hash values of VMs. Thus, the hash value can be used as the ID of VMs. The second challenge is the way to protect the ID of a storage device. If the ID of a storage device can be changed by softwares, the storage device can be allocated to an arbitrary VM. In the proposed system, the ID of a storage device is managed by the storage device hardware. With a limited interface of the storage device, the ID of the storage device can be changed only after the all contents of the storage device are deleted. Therefore, the storage device cannot be allocated to an arbitrary VM with its original contents. The final challenge is the way to communicate between H-SVM+ and a storage device. To know the ID of a storage device, H-SVM+ should be able to communicate to the storage device. However, since H-SVM+ exists within a processor-memory complex, it cannot directly communicate to the storage device. In the proposed system, the communication channel between H-SVM+ and a storage device is established by manipulating IOMMUPT, and the ID of storage device is delivered via main memory.

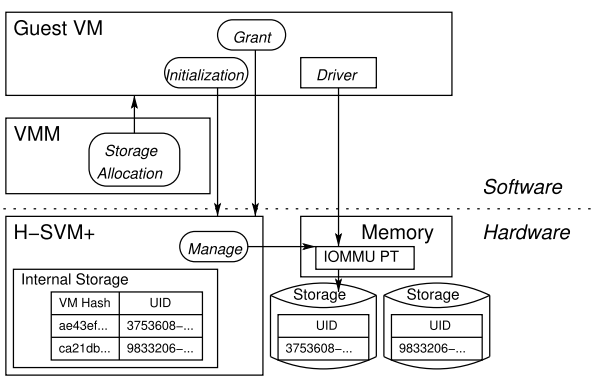



**Fig. 3** Overall Architecture

Before explaining detailed processes of the proposed system, we explain two distinct IDs for each storage. The first ID is a storage device ID (devID) which is used for identifying storage devices by VMM. Since devID is used when VMM allocates a specific storage to a guest VM, devID is represented by BDF (Bus:Device.Function) notation [11]. The second ID is a storage unique ID (UID) which is used for identifying storage devices by H-SVM+. Since UID is used only between H-SVM+ and storage devices, software components such as VMM cannot know the UID.

H-SVM+ achieves storage isolation with the two IDs as following two phases. The first phase is the initialization phase where a storage device is initialized and the UID of the storage device is registered in H-SVM+. The second phase is the grant phase where H-SVM+ grants access from a VM to a storage device. Specifically, the details of the initialization phase are as follows. The initialization process in a guest OS calls `SetupNewStorage(devID)` in H-SVM+. H-SVM+ creates a UID and stores the UID with a VM hash value in the internal storage in H-SVM+. The VM hash value is the measured hash value of the first partition and is obtained by H-SVM before a guest VM starts. H-SVM+ also writes the UID to a pre-defined area in main memory. The address of the area is known to both H-SVM+ and a storage device. H-SVM+ sets IOMMU page table to enable the storage device to access the UID. Afterwards, the initialization process in the guest OS calls `AllocateNewUID()` in the storage device. The storage device would delete all stored data in the storage and access the UID created by H-SVM+ via direct memory access (DMA) capability. Since the IOMMU page table is set by H-SVM+, the storage device can access the UID directly. The storage device stores the UID in internal meta data area. The initialization process is performed once when a storage device is initialized.

After that, the grant phase is performed to grant access from a guest OS to a storage device. Whenever a guest OS boots, the grant phase should be performed. Otherwise, the guest OS cannot access the storage device. The details of the grant phase are as follows. The grant process in a guest OS calls `GrantSetup(devID)` in H-SVM+. H-SVM+ sets the IOMMU page table for a storage device. After that, the grant process calls `ShowUID()` in a storage device via a storage device driver. The storage device writes UID to a pre-defined area with DMA. Finally, the grant process calls `GrantStorage(devID)`. After checking devID and UID in the internal storage, H-SVM+ sets IOMMU page table to enable a guest OS to access the storage device.

## 4. Evaluation

### 4.1 Security Analysis

**Confidentiality and integrity.** To guarantee the confidentiality and integrity of storage devices, a storage device

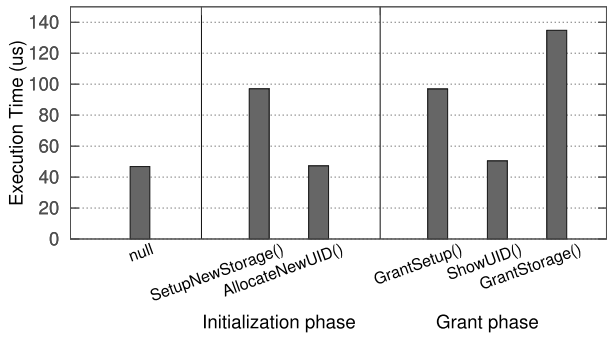



**Fig. 4** Overheads of H-SVM+

should be allocated only to the VM where the storage device is initialized. Since H-SVM+ manages a mapping table containing VM hashes and UIDs, the table cannot be altered by software components. Moreover, storage devices cannot be used without the grant phase since the accesses from a guest OS are blocked by the IOMMU page table. Thus, it is guaranteed that a storage device is always allocated to the same VM.

**Availability.** H-SVM+ does not support the availability of storage devices. Since VMM is in charge of the allocation of storage devices, VMM can deny the allocation of storage devices. Such a problem, however, is easily detected by a cloud user because the storage device is unavailable to the cloud user. Therefore, the availability issues can be mediated by service level agreement (SLA) between a cloud provider and a cloud user.

### 4.2 Performance Evaluation

Since H-SVM+ grants access permissions from VMs to storage devices, the overheads by encryption and decryption are disappeared, and thus, the performance gains are expected to be the same as the native system in Fig. 2. Therefore, in this section, we focus on the overheads caused by initialization and grant operations.

We used Xen 4.0.1 as VMM and Ubuntu 10.04 as a management VM and guest VMs. Since H-SVM+ is proposed as a hardware component, H-SVM+ should have a higher privilege than VMM. Thus, we implemented the operations of H-SVM+ and a storage device in system management mode (SMM). SMM is a special mode in x86 architecture. When a system mode becomes SMM, all software stops and only SMM code can be executed. Moreover, SMM manages a special part of main memory called system management RAM (SMRAM). Since SMRAM can be accessed only in SMM, the contents of SMRAM can be protected. Therefore, it is possible to implement a component having a higher privilege than VMM. Even though SMM is originally designed for urgent system management such as power failure, SMM is used for various implementations [12], [13] due to its capability.

We measured the execution times of each operation as depicted in Fig. 4. The experiment runs each operation 100 times, and the execution times are averaged. To enter SMM, context switch overhead is needed. By doing nothing in SMM, we measured the overhead denoted by `null`. The initialization phase takes 144$\mu$s. `SetupNewStorage()` takes a relatively long time compared to `AllocateNewUID()` because of page table allocation. Note that the time for deleting the contents of storage devices is excluded in `AllocateNewUID()` because it varies according to the performance and capacity of storage devices. The grant phase takes 282$\mu$s. `GrantStroage()` takes half of the time because setup page tables are deleted and new page tables are allocated. These overheads are relatively small compared to the overheads of cryptographic operations. Moreover, the initialization phase is performed only once when a new storage device is allocated and the grant phase is performed whenever after a guest VM boots. Therefore, the proposed system is able to isolate storage devices with a negligible overhead.

## 5. Conclusions and Limitation

Providing a secure cloud environment is the steppingstone to invigorate cloud growth. Even though the hardware-based memory isolation scheme issued a new direction, cloud users are responsible for protecting cloud storage. By isolating the storage of cloud users from a software layer completely, the proposed system not only provides a secure storage environment but also solve the overheads of encryption and decryption.

A limitation of this study is that the proposed system does not provide full cloud features such as a live migration because the data protection mechanism during a migration process does not presented. Moreover, the proposed system supports only directly assignable storage such as NVMe since the communication path between a guest VM and a storage device are protected.

### References

[1] S. Jin, J. Ahn, S. Cha, and J. Huh, "Architectural Support for Secure Virtualization Under a Vulnerable Hypervisor," Proc. 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.272–283, 2011.
[2] J. Szefer and R.B. Lee, "Architectural Support for Hypervisor-secure Virtualization," Proc. Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp.437–450, 2012.
[3] S. Jin, J. Ahn, J. Seol, S. Cha, J. Huh, and S. Maeng, "H-SVM: Hardware-assisted Secure Virtual Machines under a Vulnerable Hypervisor," IEEE Trans. Comput., [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TC.2015.2389792
[4] D.G. Murray, G. Milos, and S. Hand, "Improving Xen security through disaggregation," Proc. 4th ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE), pp.151–160, 2008.
[5] M. Halcrow, "eCryptfs: a stacked cryptographic filesystem," Linux Journal, 2007.
[6] "Blogbench," http://www.pureftpd.org/project/blogbench.
[7] "DBENCH," https://dbench.samba.org/
[8] "bonnie++," http://www.coker.com.au/bonnie++/

[9] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," in 2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), pp.1–10, Jan. 2010.

[10] J.J. Hung, K. Bu, Z.L. Sun, J.T. Diao, and J.B. Liu, "PCI Express-Based NVMe Solid State Disk," Applied Mechanics and Materials, vol.464, pp.365–368, Nov. 2013.

[11] T. Shanley and D. Anderson, PCI System Architecture, 4th Edition, MindShare, 1999.

[12] A.M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N.C. Skalsky, "HyperSentry: enabling stealthy in-context measurement of hypervisor integrity," Proc. 17th ACM conference on Computer and communications security (CCS), pp.38–49, Oct. 2010.

[13] A.M. Azab, P. Ning, and X. Zhang, "SICE: A Hardware-Level Strongly Isolated Computing Environment for x86 Multi-core Platform," Proc. 18th ACM conference on Computer and communications security (CCS), pp.375–388, Oct. 2011.