Optimized Memory Encryption for VMs Across Multiple Hosts

Optimized Memory Encryption for VMs across Multiple Hosts

Shuhei Horio, Kouta Takahashi, Kenichi Kourai, and Lukman Ab. Rahim

Abstract Recently, virtual machines (VMs) with a large amount of memory are widely used. It is often not easy to migrate such a large-memory VM because VM migration requires one large destination host. To address this issue, split migration divides the memory of a VM into small pieces and transfers them to multiple destination hosts. The migrated VM exchanges its memory data between the hosts using remote paging. To prevent information leakage from the memory data in an untrusted environment, memory encryption can be used. However, encryption overhead largely affects the performance of the hosts and the VM. This paper proposes *SEmigrate* for optimizing the memory encryption in split migration and remote paging. SEmigrate avoids decrypting memory data at most of the destination hosts to reduce the overhead and completely prevent information leakage. Also, it selectively encrypts only the memory data containing sensitive information by analyzing the memory of the guest operating system in a VM. SEmigrate could reduce the CPU utilization during encrypted split migration by 6–20% point and improve the performance of the migrated VM with encrypted remote paging to 1.9x.

Shuhei Horio
Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka, Japan, e-mail: horio@ksl.ci.kyutech.ac.jp

Kouta Takahashi
Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka, Japan, e-mail: takahashi@ksl.ci.kyutech.ac.jp

Kenichi Kourai
Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka, Japan, e-mail: kourai@ksl.ci.kyutech.ac.jp

Lukman Ab. Rahim
Universiti Teknologi Petronas, 32610 Seri Iskandar, Perak Darul Ridzuan, Malaysia, e-mail: lukmanrahim@utp.edu.my

1 Introduction

Recently, virtual machines (VMs) with a large amount of memory are widely used. For example, Amazon EC2 provides VMs with 24 TB of memory [1]. Upon host maintenance, a VM can be moved to another host using VM migration without disrupting the services provided by the VM. For large-memory VMs, however, it is not cost-efficient to always preserve large hosts with a sufficient amount of memory as the destination of occasional VM migration. These hosts cannot run other VMs even while they are not used for VM migration.

To make the migration of such large-memory VMs more flexible, split migration has been proposed [7]. Split migration divides the memory of a VM into small pieces and transfers them to multiple smaller destination hosts, which consist of one main host and one or more sub-hosts. It transfers likely accessed memory data to the main host as well as the state of the VM core such as virtual CPUs. The rest of the memory is transferred to sub-hosts. After split migration, the main host runs the VM core, while the sub-hosts provide the memory to the VM. When the VM accesses the memory existing in a sub-host, it exchanges memory data between the main host and the sub-host using remote paging. The sub-host transfers the required memory data to the main host, while the main host transfers unnecessary memory data to the sub-host.

However, it is possible to eavesdrop on the memory data of a VM during split migration and remote paging in an untrusted execution environment. For example, information leakage easily occurs if the memory data is transferred via untrusted networks. If the administrators of some of the sub-hosts are untrusted, they can eavesdrop on the memory data held in the sub-hosts. In general, the encryption of the memory data can prevent such information leakage. However, encryption overhead largely affects the performance of the hosts and the VM because encryption and decryption are performed every time memory data is transferred.

To address this performance issue, this paper proposes *SEmigrate* for optimizing memory encryption in split migration and remote paging. SEmigrate avoids decrypting memory data at sub-hosts to reduce encryption overhead and completely prevent information leakage at sub-hosts. Upon split migration, it encrypts memory data at the source host and decrypts that data only at the destination main host. Upon remote paging, it encrypts or decrypts memory data only at the main host. In addition, SEmigrate selectively encrypts only memory data containing sensitive information to further reduce encryption overhead.

To obtain information needed for the selective encryption, SEmigrate analyzes the memory of the guest operating system (OS) in a VM using a technique called VM introspection (VMI) [3]. For example, it considers that the free memory of the guest OS does not contain sensitive information. If SEmigrate determines that memory data to be transferred is part of the free memory, it does not encrypt that memory data. When the user specifies that an application does not deal with sensitive information, SEmigrate does not encrypt the entire memory of all the processes executing that application.

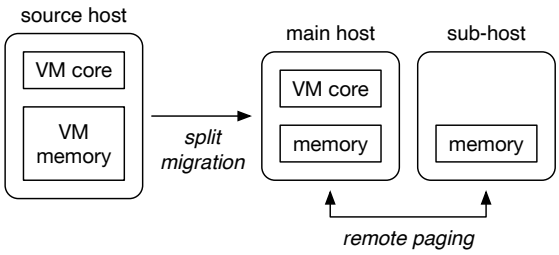


Fig. 1 Split migration.

We have implemented SEmigrate in KVM supporting split migration and remote paging. To confirm performance improvement by SEmigrate, we ran an application using a large amount of memory in a VM and examined the performance of encrypted split migration and the migrated VM. As a result, it was shown that SEmigrate could reduce the CPU utilization during encrypted split migration by 6–20% point. Also, we showed that SEmigrate could improve the performance of the migrated VM with encrypted remote paging to 1.9x.

The organization of this paper is as follows. Section 2 describes split migration and remote paging and the issues of memory encryption. Section 3 proposes SEmigrate for optimizing memory encryption in split migration and remote paging. Section 4 explains the implementation of SEmigrate and Section 5 shows our experimental results. Section 6 describes related work and Section 7 concludes this paper.

2 Memory Encryption of VMs across Hosts

2.1 Split Migration

Split migration [7] divides the memory of a VM into small pieces and transfers them to multiple small destination hosts, as illustrated in Fig. 1. The destination hosts consist of one main host and one or more sub-hosts. Split migration transfers likely accessed memory data to the main host as much as possible. It also transfers the state of the VM core such as virtual CPUs and devices. The rest of the memory data is transferred to one of the sub-hosts.

After split migration, the migrated VM runs across the main host and the sub-hosts. The main host runs the VM core, while the sub-hosts provide the memory to the VM core. When the VM core requires the memory data existing in a sub-host, that data is exchanged between the main host and the sub-host using remote paging. The sub-host transfers the memory data required by the VM core, which is called a page-in. At the same time, the main host transfers unlikely accessed memory data to the sub-host, which is called a page-out.

2.2 Encryption of Memory Data

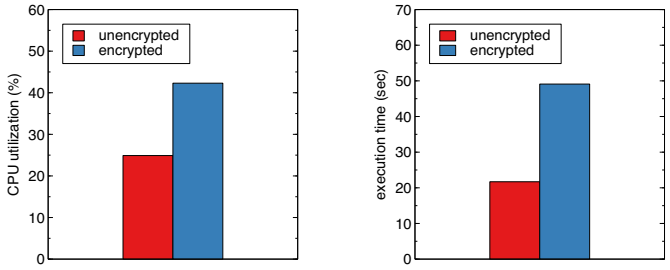
In an untrusted execution environment, it is possible to eavesdrop on the memory data of a VM transferred during split migration and remote paging. For example, information leakage easily occurs if the memory data is transferred via untrusted networks. Similarly, the memory data can be exposed if remote paging is performed between the main host and sub-hosts via untrusted networks. In addition, the memory data can be stolen by untrusted administrators at any host. Fortunately, several mechanisms have been proposed to protect the memory of a running VM [10, 4, 8]. Since we can use such memory protection mechanisms at the source host and the main host running the VM core, we assume that information leakage does not occur at these hosts in this paper.

In general, information leakage can be prevented by encrypting the memory data of a VM. Upon split migration, the memory data is encrypted at the source host by using an encrypted communication channel such as SSL. Encrypted data is transferred to the destination main host and sub-hosts and is then decrypted by the channel. After that, the sub-hosts re-encrypt the decrypted memory data to securely hold it against untrusted administrators. The reasons why decryption and re-encryption are required are that the channel automatically decrypts memory data and that it is difficult to continue to use the decryption key created for the channel after the communication. Note that such re-encryption is not necessary at the main host because the memory data is managed in a protected manner.

Upon remote paging, the memory data held in a sub-host is first decrypted. Then, it is re-encrypted at the sub-host by using an encrypted communication channel established between the sub-host and the main host. It is transferred to the main host for a page-in and is then decrypted by the channel. For a page-out, unnecessary memory data is encrypted at the main host by using this channel and is then transferred to the sub-host. It is decrypted at the sub-host and is then re-encrypted to be securely held.

However, using encrypted communication channels imposes a large overhead because encryption and decryption are performed whenever memory data is transferred. Fig. 2(a) shows the average CPU utilization at the source host during split migration of a VM. This measurement was done using the experimental setup in Section 5. Although the migration time almost did not increase by memory encryption, the CPU utilization became 1.7x higher. Fig. 2(b) shows the performance of the VM migrated by split migration. The execution time of a memory benchmark running in the VM increased to 2.2x due to memory encryption in remote paging.

In terms of security, the memory data of a VM is still exposed at sub-hosts due to re-encryption. Since it is decrypted temporarily by an encrypted communication channel, attackers can eavesdrop on the memory data before the data is re-encrypted. In addition, the memory data held in sub-hosts can be easily decrypted if its decryption keys are stolen by the administrators of sub-hosts.



(a) CPU overhead during split migration

(b) VM performance after split migration

Fig. 2 The encryption overhead during and after split migration.

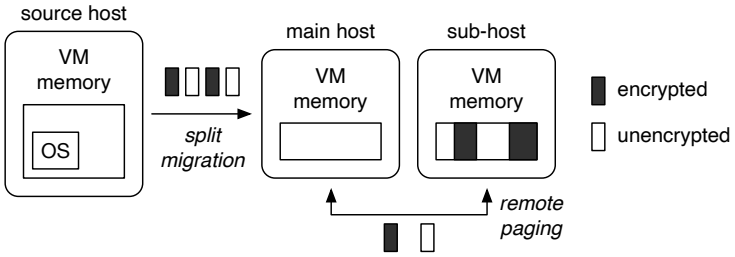


Fig. 3 The optimization of the encryption of memory data in SEmigrate.

3 SEmigrate

This paper proposes SEmigrate for optimizing memory encryption in split migration and remote paging. As illustrated in Fig. 3, SEmigrate avoids decrypting the memory data of a VM at sub-hosts to reduce encryption overhead and completely prevent information leakage. To further reduce the overhead, it selectively encrypts only the memory that contains sensitive information. To use memory attributes and process information for these optimizations, it analyzes the memory of the guest OS in a VM using VMI [3].

3.1 No Decryption at Sub-hosts

SEmigrate avoids the decryption of the memory data of a VM at sub-hosts. Upon split migration, the source host encrypts memory data, while only the destination main host decrypts it. The destination sub-hosts hold it without decrypting it. To enable decrypting the encrypted memory data later, SEmigrate uses an encryption key that is available through the life cycle of a VM. Therefore, it does not use an

encrypted communication channel that is established only for VM migration. This can reduce the overhead of the re-encryption of memory data as well as that of the decryption. Also, this can prevent information leakage by temporarily decrypting memory data in an encrypted communication channel. Since the sub-hosts do not manage the decryption keys of the encrypted memory data, even the administrators of the sub-hosts cannot decrypt or eavesdrop on the memory data.

Upon remote paging, SEmigrate encrypts and decrypts the memory data of a VM only at the main host. For a page-in, a sub-host does not decrypt the memory data requested by the main host. Then, it can securely transfer encrypted memory data to the main host without using an encrypted communication channel. The main host decrypts the received memory data and uses it. For a page-out, the main host encrypts unnecessary memory data without using an encrypted communication channel and then securely transfers it to a sub-host. The sub-host holds it without decrypting it. SEmigrate uses the encryption and decryption keys shared in split migration to encrypt and decrypt memory data at the main host in remote paging.

3.2 Selective Encryption Based on in-VM Information

SEmigrate selectively encrypts the memory data of a VM at the source host and the main host to reduce encryption overhead. Upon split migration, the source host encrypts only the memory data that contains sensitive information, while it does not encrypt the other memory data. The destination main host decrypts the received data only if the memory data is encrypted. The destination sub-hosts hold it without encrypting it even if the memory data is not encrypted. Since the memory data does not contain sensitive information, it does not need encryption at sub-hosts as well.

Upon remote paging, a sub-host transfers the memory data requested for a page-in to the main host without encrypting it even if the memory data is not encrypted. The main host decrypts the received data only if the memory data is encrypted. For a page-out, the main host transfers unnecessary memory data to a sub-host without encrypting it if the memory data does not contain sensitive information. The sub-host holds it without encrypting it even if the memory data is not encrypted.

SEmigrate considers various memory regions of the guest OS in a VM to be not sensitive. For example, the memory regions that are not used by the guest OS do not contain sensitive information. Therefore, SEmigrate does not encrypt such free memory. When memory data is transferred, SEmigrate obtains its memory attribute from the guest OS using VMI and examines whether it is free memory or not. Since sensitive information may be left in free memory after used memory is released, SEmigrate transfers zero-filled data instead of the actual data of free memory.

When the user specifies an application that does not deal with sensitive information in a VM, SEmigrate does not encrypt the memory of the corresponding processes used to execute that application. For example, if an in-memory database deals with only encrypted data, its memory data does not need to be further encrypted when it is transferred. When memory data is transferred, SEmigrate finds

the memory region to which that data belongs and identifies the process that owns that memory region. If the name of that process is equal to the specified one, SEmigrate does not encrypt the memory data to be transferred.

4 Implementation

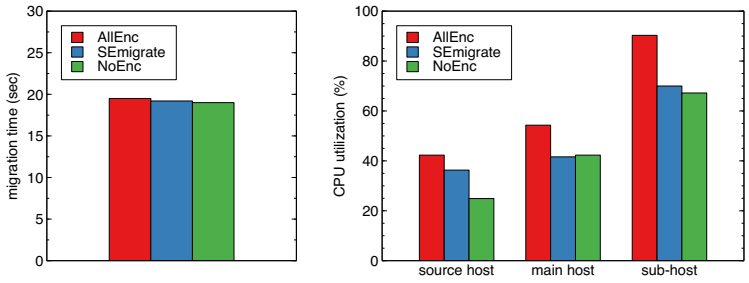
We have implemented SEmigrate in QEMU-KVM 2.11.2 that supports split migration and remote paging. We assume Linux 4.18 as a guest OS to apply VMI for selective encryption, but we can use the other versions of Linux. We used AES-ECB with AES-NI in OpenSSL and a 128-bit key for memory encryption. To transparently analyze and obtain information on the guest OS using VMI, we have ported the LLView framework [5] to QEMU-KVM. LLView enables writing programs for VMI using the source code of the Linux kernel. It transforms a written program at compile time so that the program accesses the memory of a VM when it accesses the data of the guest OS.

When SEmigrate transfers a 4-KB memory page in a VM, it checks whether that page is included in a free memory region. The Linux kernel allocates contiguous 2^n physical pages at once using the buddy system. Also, it manages free memory pages as free memory regions that consist of 2^n pages. SEmigrate analyzes the memory of the VM using VMI and finds the page structure corresponding to the target page in the Linux kernel. On the basis of the information stored in that page structure, it can determine that the target page is free.

SEmigrate also checks whether a page to be transferred is part of the specified application process. For this purpose, it finds the process that owns the target page. First, it finds the page structure corresponding to the target page and obtains the index in the red-black tree used for managing virtual memory areas from that page structure. Using this index, SEmigrate searches the red-black tree and finds the target virtual memory area. Next, it finds the process owning that area and obtains its process name from the task_struct structure. If the process name is equal to the specified one, SEmigrate determines that the target page is part of the process memory that should not be encrypted.

5 Experiments

We conducted several experiments to examine performance improvement by SEmigrate in encrypted split migration and remote paging. For comparison, we used two methods that encrypted all the memory data (AllEnc) and that encrypted no memory data (NoEnc). For the source host and the destination main host, we used two PCs with an Intel Core i7-8700 processor, 64 GB of memory, and 10 Gigabit Ethernet (GbE). For the destination sub-host, we used a PC with an Intel Xeon E3-1226 v3 processor, 32 GB of memory, and 10 GbE. We created a VM with one virtual CPU



(a) Migration time (b) CPU utilization

Fig. 4 The performance of split migration.

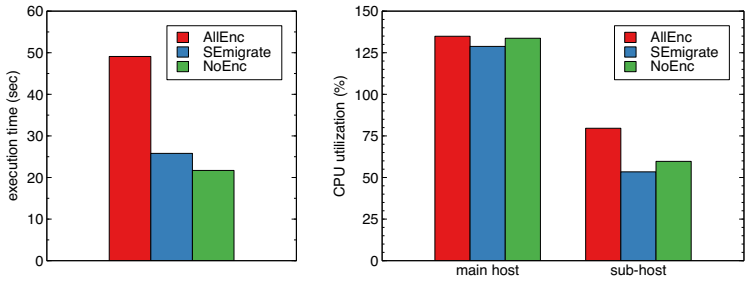
and 20 GB of memory. In all of the PCs and the VM, we ran Linux 4.18. Upon split migration, we equally divided the memory of the VM.

To examine performance improvement in split migration, we ran an application using 10 GB of memory in the VM and specified that its memory did not need to be encrypted. As shown in Fig. 4(a), the migration time was almost unchanged between the three. However, SEmigrate could reduce the CPU utilization during split migration. Fig. 4(b) shows that the reduction was 6% point, 13% point, and 20 % point at the source host, the destination main host, and the destination sub-host, respectively. The CPU utilization in SEmigrate was almost the same as in NoEnc at the destination hosts. This is because the destination hosts did not decrypt most of the memory data by the optimization of memory encryption. In contrast, the CPU utilization was higher than NoEnc at the source host probably due to the overhead of selective encryption.

To examine the performance of the VM after split migration, we measured the execution time of a memory benchmark running in the VM. This benchmark allocated 10 GB of memory and repeatedly wrote one byte per page to cause a lot of remote paging. As shown in Fig. 5(a), SEmigrate could reduce the execution time by 47% thanks to the optimization of memory encryption in remote paging. The performance was comparable to that of NoEnc. Fig. 5(b) shows the CPU utilization at the main host and the sub-host. It was almost unchanged at the main host, whereas SEmigrate could reduce it by 26% point at the sub-host. This is because the sub-host avoided encryption and decryption completely.

6 Related Work

For traditional VM migration, various optimizations using VMI have been proposed. MiG [6] obtains memory attributes from the guest OS in a VM and optimizes the compression algorithm for memory data using the obtained information. For exam-



(a) Execution time (b) CPU utilization

Fig. 5 The performance of a memory benchmark in the migrated VM.

ple, it transfers only page frame numbers for free memory. It compresses the heap area using gzip because of its high redundancy. Such optimizations can reduce the amount of transferred memory by 51–61% and halve the migration time. However, MiG first saves all the states of a VM and then compresses the memory data. Therefore, it does not support live migration, which migrates a VM without stopping it, unlike SEmigrate.

IntroMigrate [2] identifies free memory of the guest OS in a VM and avoids transferring it to shorten the migration time. It obtains information on the entire free memory at the beginning of VM migration. Therefore, it can apply the optimization based on old information for the memory that becomes in use or free during the migration. It relies on the re-transfer mechanism to transfer the memory data that became in use. Using old information can cause information leakage in the optimization of memory encryption. Similar work [9] avoids transferring the page cache of the guest OS as well as free memory.

The secure virtualization architecture [4] and VMCrypt [8] prevent information leakage from the memory of a running VM. They provide the unencrypted version of the memory to the VM, while they provide the encrypted version to the management VM used by the administrators. Upon VM migration, they obtain and transfer the encrypted memory of a VM in the management VM. SEmigrate can apply such memory protection mechanisms to the source host and the destination main host to prevent information leakage unless the hypervisor is compromised. CloudVisor [10] can protect the memory of a VM without relying even on the hypervisor by running the security monitor under the hypervisor.

7 Conclusion

This paper proposed SEmigrate for optimizing memory encryption in split migration and remote paging. SEmigrate avoids decrypting the memory data of a VM

to reduce encryption overhead and completely prevent information leakage at subhosts. In addition, it selectively encrypts only the memory data that contains sensitive information to further reduce encryption overhead. To enable this, it analyzes the memory of the guest OS using VMI and its applications in a VM and identifies free memory and specified applications. We have implemented SEmigrate in KVM and showed that SEmigrate could reduce the CPU utilization during encrypted split migration by 6–20% point. Also, it could improve the performance of the migrated VM with encrypted remote paging to 1.9x.

One of our future work is to extend the target of our selective encryption to other memory regions, e.g., the code area and specified data in applications. To use application-specific information, SEmigrate needs to analyze the memory of applications as well as that of the OS kernel. After that, we are planning to apply SEmigrate to real applications and confirm that SEmigrate can perform selective encryption using application-specific information. Then, we will show the performance improvement in encrypted split migration and remote paging.

Acknowledgements The research results have been achieved by the "Resilient Edge Cloud Designed Network (19304)," the Commissioned Research of National Institute of Information and Communications Technology (NICT), Japan.

References

1. Amazon Web Services, Inc.: Amazon EC2 High Memory Instances. https://aws.amazon.com/ec2/instance-types/high-memory/ (2019)
2. Chiang, J., Li, H., Chiueh, T.: Introspection-based Memory De-duplication and Migration. In: Proceedings of ACM International Conference Virtual Execution Environments, pp. 51–62 (2013)
3. Garfinkel, T., Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection. In: Proceedings of Network and Distributed Systems Security Symposium, pp. 191–206 (2003)
4. Li, C., Raghunathan, A., Jha, N.: A Trusted Virtual Machine in an Untrusted Management Environment. IEEE Trans. Services Computing **5**(4), 472–483 (2012)
5. Ozaki, Y., Kanamoto, S., Yamamoto, H., Kourai, K.: Detecting System Failures with GPUs and LLVM. In: Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems, pp. 47–53 (2019)
6. Rai, A., Ramjee, R., Anand, A., Padmanabhan, V., Varghese, G.: MiG: Efficient Migration of Desktop VMs Using Semantic Compression. In: Proceedings of USENIX Annual Technical Conference, pp. 25–36 (2013)
7. Suetake, M., Kashiwagi, T., Kizu, H., Kourai, K.: S-memV: Split Migration of Large-memory Virtual Machines in IaaS Clouds. In: Proceedings of IEEE International Conference Cloud Computing, pp. 285–293 (2018)
8. Tadokoro, H., Kourai, K., Chiba, S.: Preventing Information Leakage from Virtual Machines' Memory in IaaS Clouds. IPSJ Online Transactions **5**, 156–166 (2012)
9. Wang, C., Hao, Z., Cui, L., Zhang, X., Yun, X.: Introspection-based Memory Pruning for Live VM Migration. International J. Parallel Program **45**(6), 1298–1309 (2017)
10. Zhang, F., Chen, J., Chen, H., Zang, B.: CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In: Proceedings of ACM Symposium Operating Systems Principles, pp. 203–216 (2011)