Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

eMotion: An SGX extension for migrating enclaves

Jaemin Park[a,b,*], *Sungjin Park*[a], *Brent Byunghoon Kang*[b], *Kwangjo Kim*[b]

[a] *The Affiliated Institute of ETRI, P.O.Box 1, Yuseong, Daejeon, Republic of Korea*
[b] *Graduate School of Information Security, School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong, Daejeon, Republic of Korea*

ARTICLE INFO

Article history:
Received 18 August 2017
Revised 30 May 2018
Accepted 26 September 2018
Available online 4 October 2018

Keywords:
SGX
Enclave migration
Managed migration
OpenSGX
Trusted execution environment

ABSTRACT

Software Guard Extensions (SGX) is a good candidate to address sensitive information disclosure in cloud computing because SGX creates enclaves for applications that protect security sensitive code and data from malicious access. However, existing SGX-enabled Virtual Machine Managers (VMMs) do not provide live migration of SGX-enabled Virtual Machines (VMs). This management operation is impossible because the VMM cannot directly access the Enclave Page Cache (EPC) pages where the VM's enclaves reside. SGX supports the EPC page swapping mechanism that evicts the EPC pages into the untrusted memory which the VMM can access. However, this mechanism has the limitations to be applied to enclave migration. In this paper, we propose an SGX extension for migrating enclaves called eMotion that adds additional instructions and migration support to the SGX architecture for enabling the secure managed migration of running enclaves. eMotion allows that the participating hosts establish a key used in enclave migration and the VMMs in the hosts migrate running enclaves using the established key. We implement a prototype on top of OpenSGX, an open source SGX emulator, to demonstrate the operations of eMotion and to estimate the impact on enclave migration.

1. Introduction

Though cloud computing has been widely adopted in various areas, security administrators are still reluctant to introduce cloud computing into their own organizations due to various security concerns, such as information disclosure. First, in a multi-tenant environment, attackers could coincidently access sensitive information that resides in other guest Virtual Machines (VMs) due to hypervisor vulnerabilities like CVE-2015-3340. Second, a malicious insider can intentionally access sensitive information owned by a victim organization's VM with cloud management operations. For example, the Virtual Machine Manager (VMM) takes full control of all guest VMs and the cloud administrator manages the cloud infrastructure using the VMM. Therefore, if the cloud administrators attempt to obtain sensitive data used in the guest VMs, they can achieve it with operations for cloud management. (e.g., taking a VM snapshot containing a cryptographic key for database encryption.)

Software Guard Extensions (SGX) is a good candidate that addresses information disclosure in cloud computing. SGX creates *enclaves* for applications that protect security sensitive code and data from malicious access. The enclave pages consist of the contents of the enclave and the associated data structures (i.e., SGX Enclave Control Structure (SECS), and Thread Control Structure (TCS)), and are stored in the Enclave Page Cache (EPC). The EPC is a subset of the Pro-

[*] Corresponding author.
E-mail address: jmpark@nsr.re.kr (J. Park).

https://doi.org/10.1016/j.cose.2018.09.008
0167-4048/© 2018 Elsevier Ltd. All rights reserved.

cessor Reserved Memory (PRM), which is a part of DRAM invisible to other software. This feature prevents higher privileged software (e.g., operating systems (OSes), VMMs, etc) from accessing sensitive information in the enclave. Therefore, the VMM cannot notice the enclave pages even if it has information leak vulnerabilities. Because CPU fetches the contents of the enclave from the PRM in an encrypted form, the enclave can be protected from external access as well as from probing attacks on the DRAM bus by an insider attacker.

However, there is still a challenging problem with imposing SGX into cloud computing: *the existing SGX-enabled VMMs (Intel Corporation, 2016a) do not provide live migration for SGX-enabled VMs*. Generally, in the managed migration of the VM, the source VMM transfers the entire VM's memory pages to the destination VMM until the VMs in the different physical machines, the source and destination hosts, are consistent. Then, the destination VMM starts the migrated VM, and the source VMM stops its VM. To this end, for managed live migration of an SGX-enabled VM, the VMM should transfer the enclave pages to the destination host.

However, the VMM cannot transfer the enclave pages as usual because SGX prevents the VMM, one of higher privileged software, from accessing directly the PRM as mentioned above. SGX Developer Guide (Intel Corporation, 2016b) provides the guideline for migrating *enclave data* across the platforms, but this guideline cannot be applied to migration of other enclave pages excluding the enclave data. Intel's patents (Rozas et al., 2017; 2018) presented the instructions and the platform for enclave migration, but the practical implementation is not realized yet and the key establishment for enclave migration is still conceptual.

As a realized mechanism, Gu et al. (2017) presented a secure enclave migration in a self-migration manner. However, the source host cannot migrate enclaves, which do not use a specific library for enclave migration, to the destination host. Thus, this constraint can cause a decline of usability because enclave developers should re-implement the existing enclaves.

The EPC page swapping mechanism can also be considered to enable the OS/VMM to evict EPC pages into the untrusted memory and load them into EPC later using dedicated instructions. However, it is infeasible to apply this mechanism to managed enclave migration. First, the destination host cannot decrypt the evicted enclave pages because a key used in this mechanism is unique and cannot leave the processor. Second, this mechanism cannot evict some EPC pages for data structures such as TCS. Third, this mechanism cannot migrate the running enclave because the eviction is only applicable to the stopped enclave.

In this paper, we propose an SGX extension for migrating enclaves called *eMotion* that adds additional instructions and migration support for enabling the secure managed migration of running enclaves. eMotion allows that the different physical hosts establish a key used in enclave migration securely and the VMMs in the hosts migrate running enclaves using the established key. eMotion guarantees that only the designated enclave and the SGX-enabled processor can access this key.

The followings are the contributions of our paper.

- *SGX extension for migrating enclaves*. We supplement the current SGX implementation with eMotion, additional instructions and migration support in order that the VMM migrates the running enclaves securely between the different physical hosts.
- *Architecture for migrating enclaves*. We present an architecture to show the practical deployment of eMotion.
- *Prototype implementation*. We implement a prototype on top of OpenSGX(Jain et al., 2016), an open source SGX emulator, to demonstrate the operations of eMotion and to estimate the impact on enclave migration.

This paper is organized as follows: Section 2 describes the SGX features for managed enclave migration. We propose eMotion in Section 4 and present a prototype based on OpenSGX in Section 5. We evaluate eMotion and its prototype in Section 6. We discuss limitations and future work in Section 7. We mention related work, and conclude this paper in Section 8 and 9, respectively.

2. Software Guard Extensions (SGX)

SGX is an extended set of instructions that supports *enclaves* where security sensitive code and data are protected by an SGX-enabled processor. The SGX-enabled processor guarantees the confidentiality and the integrity of an enclave by using an isolated memory area, the EPC, that cannot be accessed from outside the enclave. When the enclave is loaded and initialized, the SGX platform detects if the enclave is not altered by comparing the enclave's calculated measurement with the pre-produced one. Remote attestation allows a remote entity to verify that the enclave is running inside the SGX-enabled processor and thus can be trustworthy. In this section, we explain SGX features used in eMotion and the SGX details can be found in Intel Corporation (2016b, 2014), Costan and Devadas (2016), and Anati et al. (2013).

2.1. *SGX data structures*

SGX data structures are a collection of data structures used to manage enclave operations. The EPC, a subset of the PRM, stores these data structures along with the contents of the enclaves. The SGX-enabled processor records the metadata of each EPC page in the Enclave Page Cache Map (EPCM). In this section, we briefly explain the structures used in eMotion.

SGX Enclave Control Structure (SECS) is metadata associated with each enclave. A dedicated EPC page, PT_SECS, stores SECS. SECS is allocated when an enclave is created and deallocated when the enclave is destroyed. This structure contains information of enclave identification (ID), enclave measurement, and enclave control. SECS identifies an enclave inside and outside the processor.

Thread Control Structure (TCS) is metadata used to support the multi-thread execution of an enclave code. A dedicated EPC page, PT_TCS, stores TCS. Each logical processor uses TCS to execute the enclave code.

PAGEINFO (Page Information) is an architectural data structure used as a parameter in EPC management instructions. This data structure contains the addresses of the enclave page, SECINFO (Security Information)/PCMD (Page Crypto Metadata), and SECS. SECINFO consists of flags that describe the state of the enclave page. PCMD is crypto metadata associated with a paged-out EPC page that includes enclave ID, MAC (Message Authentication Code) for the evicted EPC page, page metadata, etc.

2.2. *Local attestation*

Local attestation is a cryptographic way for internal enclaves to attest other enclaves that reside inside the processor for providing higher-level functions like remote attestation. An enclave can prove its identity to other enclaves by producing REPORT because the signature block in the REPORT is produced by the same platform and thus is verifiable inside the processor.

2.3. *Remote attestation*

Remote attestation is a cryptographic way for remote entities to attest to the trustworthiness of the underlying hardware platform and the running enclaves. Intel provides the enclave for remote attestation, Quoting Enclave (QE), as an Architectural Enclave (AE) that is the privileged enclave in the SGX framework. QE produces QUOTE and the remote entities verify the signature block in the QUOTE by using the public key from the Intel Enhanced Privacy ID (EPID) group key.

2.4. *EPC page swapping*

The BIOS sets the size of the PRM, and thus SGX supports EPC page swapping for the OS/VMM to evict EPC pages into untrusted memory in order to overcome the limited size of the PRM. EPA allocates a version array where random numbers used to encrypt each EPC page for the anti-replay are stored. EWB evicts EPC pages with encryption and integrity protection, and ELDU/B loads them with integrity check and decryption. Thus, EPC paging instructions can maintain the same security properties (confidentiality, anti-replay, and integrity) with the PRM. The key used in this mechanism is unique for the specific processor and the outside of the processor cannot access this key. Prior to the eviction, EWB assures that the EPC pages have been blocked and the running enclave is stopped.

3. Problem definition

3.1. *System models*

We consider two different physical machines, the source host (H_S) and the destination host (H_D) where the source VMM (VMM_S) and the destination VMM (VMM_D) are running. In general, the VMM supports the migration of multiple VMs concurrently. For the simplicity and focusing on the intrinsic problems of enclave migration, we assume that H_S has a single VM (V) which uses enclaves (E) including AEs and E has no sealed data. VMM_S executes a migration protocol P to transfer the memory pages of V and E along with the VM state (S) to VMM_D. VMM_D also executes P to load the memory pages of V and E along with S into its memory for restoring V and E. H_S and H_D locate in the same network infrastructure where they can communicate each other and access the same VM image of V.

3.2. *Threat models*

We trust only enclaves and the mechanisms implemented in the SGX-enabled processors. VMM_S and VMM_D are usually operating as intended, but they can become *malicious* when the attacker *Adv*. corrupts them via the security attacks or software bugs of the VMMs and/or the VMs. As a result, *Adv*. can have full control over the memory and the network resources of the VMMs and the VMs, and it can read all memory pages of DRAM excluding the PRM and sniff all network packets. When VMM_S and VMM_D execute *P*, *Adv*. attempts to acquire the valuable information of *E* by sniffing the transferred memory pages and reading the memory pages that two VMMs can access.

Adv. can conduct the attacks like rollback and forking attacks (Brandenburger et al., 2017) that can violate the data consistency of *E*. *Adv*. that subverts VMM_S can also incur the state inconsistency of *E* by preventing VMM_S from tracking the changes of the EPC pages during *P*. We do not consider these types of attacks because cloud tenants can detect them via the existing detection mechanism (Brandenburger et al., 2017). *Adv*. can simply discard messages for executing *P*; such a denial-of-service (DoS) attack is out of scope in this paper.

3.3. *Goals*

To migrate *E* in the managed manner securely, two different physical hosts (i.e., H_S and H_D) should establish *Migration Master Key* (*MMK*). Also VMM_S and VMM_D can use *MMK* to migrate enclave pages of *E*. We define the goals of these operations for secure managed migration of the enclave.

G1: End-to-end protection on migrated enclave pages
H_S and H_D should establish *MMK* without the involvement of an additional server (e.g., trusted third party). This *MMK* should protect the migrated enclave pages between H_S and H_D in the end-to-end manner. When evicted to the untrusted memory by VMM_S, transferred to VMM_D by VMM_S, and loaded to the PRM by VMM_D, the migrated enclave pages should retain Confidentiality, Integrity, and Anti-replay (CIA) not to ruin the genuine security properties supported by SGX. This end-to-end protection prevents *Adv*. from extorting the sensitive information via sniffing the migrated enclave pages between two hosts or reading the untrusted memory in both hosts.

G2: Restricted access to *MMK*
Only the designated enclave and the SGX-enabled processor should be able to access *MMK*. It is crucial to restrict the access on *MMK* only to trustworthy parties because *Adv*. can attempt to steal *MMK* by reading the untrusted memory in both hosts. If other enclaves except for the designated enclave are malicious or erroneous, *Adv*. can use these enclaves to export *MMK* to the untrusted memory.

Thus, *MMK* should not be revealed to *V*, VMM_S, VMM_D and other software components except for the designated enclave.

G3: New SGX instructions for VMMs to migrate running enclaves
New SGX instructions should support VMMs to migrate running enclaves because VMMs cope with operations for migrating the SGX-enabled VM (*V* and *E*) in the managed manner. Using these instructions, VMM_S should be able to evict the contents of the enclave as well as the associated data structures to the untrusted memory securely. VMM_D should be able to load the entire enclave pages to its PRM by leveraging these instructions. Recall that the EPC page swapping mechanism cannot evict some EPC pages (e.g., TCS) and can evict only the stopped enclave.

4. Design

4.1. *Overview*

eMotion is an SGX extension for VMMs to migrate the running enclave securely. eMotion consists of additional instructions and migration support as shown in Fig. 1. Migration Enclave (ME), a new AE, establishes *MMK* between H_S and H_D securely. eMotion adds a new SGX instruction (EPUTKEY), one SECS attribute (MIGRATION) and one register (MKR) to the SGX-enabled processor for enforcing the access control on *MMK*. eMotion also adds new SGX instructions (ESE, ESL) to the SGX-enabled processor so that VMM_S and VMM_D can migrate *E* using *MMK*.

Using eMotion, we introduce two phases: *key exchange with remote attestation* (P_1) and *secure eviction and loading for migration* (P_2), which compose *P*. ME executes P_1 to establish *MMK* via remote attestation and store *MMK* into MKR (Migration Key Register) of the SGX-enabled processor. VMM_S and VMM_D proceed P_2 to migrate *E* using *MMK*.

In this section, we explain eMotion by dividing it into two distinct extensions: one for P_1 and the other for P_2. We also present diagrams of two phases and an architecture based on eMotion.

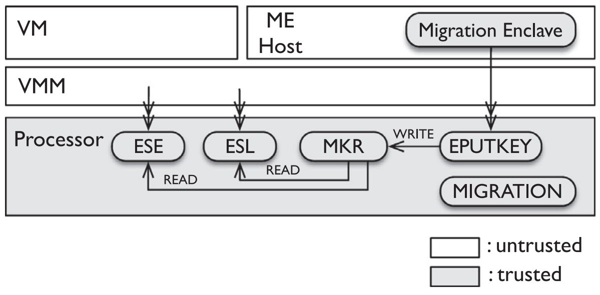


Fig. 1 – eMotion; register read/write (→) and instruction execution (↠).

4.2. *SGX extension for key exchange with remote attestation*

To migrate *E* from H_S to H_D securely, H_S and H_D should establish *MMK* first. MKR of the SGX-enabled processor should store the established *MMK* for being used in P_2. Other entities except for the designated enclave and the SGX-enabled processor should not be able to access this key.

4.2.1. *Establishing migration master key by migration enclave*

Migration Enclave (ME), which belongs to each host, is an AE that establishes *MMK* between H_S and H_D. MEs perform mutual remote attestation to convince that the SGX-enabled processor and ME in the other host are trustworthy. During the processes of remote attestation, MEs exchange keying materials like nonces and agree on *MMK* per the result of remote attestation. eMotion is independent of the underlying key exchange protocol used by ME. ME can utilize any key exchange protocols such as Diffie-Hellman (DH) key exchange protocol (Diffie and Hellman, 1976) depending on the security policy defined by the ME provider.

When enclave migration starts, the VMMs trigger ME Hosts to launch P_1 for establishing *MMK* between the source ME (ME_S) and the destination ME(ME_D), that are running in H_S and H_D, respectively. Then, ME Host executes its ME to manipulate the key exchange messages according to Algorithm 1. ME generates a key exchange message (*msg*). ME calculates the hash value, *h*, of *msg* using a cryptographic hash function, H(·). ME invokes `LocalAttest(·)` to obtain REPORT that includes a MAC tag for local attestation. `LocalAttest(·)`, which calls the EREPORT instruction, calculates the MAC of the REPORT data structure containing *h* using Report key, and feeds the MAC into the MAC tag. ME requests QE to generate QUOTE by sending *msg* and REPORT. QE checks if REPORT is valid by re-computing the MAC over the REPORT data structure using *msg* and Report key, and verifying that ME produced REPORT inside the same SGX-enabled processor. Note that the SGX implementation guarantees that Report key is known only to the target enclave (i.e., QE) and the EREPORT instruction (Anati et al., 2013). Then, QE produces QUOTE of *msg* and replies QUOTE to ME. The ME Host sends QUOTE with *msg* to the other ME Host for remote attestation.

Algorithm 1 Protocol for manipulating key exchange messages.

```
ME : Generate key exchange message, $msg$
    $h \leftarrow \mathtt{H}(msg)$
    $\mathtt{REPORT} \leftarrow \mathtt{LocalAttest}(h)$
ME→QE : $msg$, REPORT
QE: if REPORT is valid then Generate QUOTE else Abort
QE→ME : QUOTE
ME: return $msg$, QUOTE
```

ME executes Algorithm 2 to derive *MMK*. Suppose that the ME Host receives *msg′* and QUOTE′ from the other ME Host, whereas the key exchange message of the ME Host is *msg*. By verifying QUOTE′ for remote attestation, ME convinces that the opposite host equips with the legitimate SGX-enabled processor and AEs. To this end, the ME Host connects to the Intel-

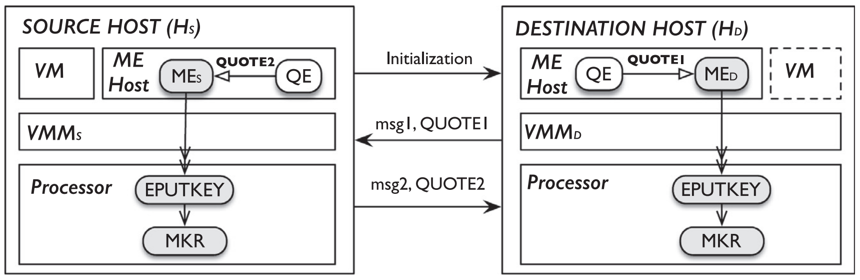


Fig. 2 – Diagram of key exchange with remote attestation; register read/write (→) and instruction execution (↠).

Algorithm 2 Protocol for deriving *MMK*.

```
1: if QUOTE′ is valid then
2:     if msg is NULL then Execute Algorithm 1
3:     $MMK \leftarrow$ Derive($msg$, $msg'$)
4: else Abort
5: return MMK
```

operated service called Intel Attestation Service (IAS) (Intel Corporation) that verifies QUOTE′ and returns the result of remote attestation via the protocol that the SGX implementation supports. If ME did not generate its key exchange message (*msg*), ME executes Algorithm 1. After remote attestation completes successfully, MEs invoke `Derive(·)` to derive *MMK* using keying materials that are exchanged via *msg* and *msg′* during remote attestation. Only MEs in H_S and H_D can access *MMK* at this point, and any additional server cannot participate in this key exchange. ME utilizes local and remote attestation that the SGX implementation supports, and thus refer to Intel Corporation (2014), Intel Corporation (2016b), Costan and Devadas (2016), Anati et al. (2013), and Intel Corporation for further details.

4.2.2. *Enforcing access control on migration master key*

EPUTKEY is an ENCLU instruction (i.e., user-level instruction) to store *MMK* into MKR of the SGX-enabled processor. MEs execute this instruction after *MMK* is established so that the SGX-enabled processors can use *MMK* to migrate *E*. For the end-to-end protection on the migrated enclave pages, *MMK* should be accessible only by MEs (i.e., the producers of *MMK*) and the SGX-enabled processors (i.e., the consumers of *MMK*).

To realize this restriction, we add an access control mechanism that utilizes Launch Enclave (LE) and MIGRATION. MIGRATION is the proposed SECS attribute that is added into the current SGX implementation in order that only ME can execute EPUTKEY. LE checks if MIGRATION of each enclave is illegally configured when the enclave is initialized. Furthermore, the SGX-enabled processor allows only the enclave whose MIGRATION is set to true to invoke EPUTKEY.

Generally, LE is an AE that prevents unauthorized enclaves from setting specialized attributes (e.g., PROVISONKEY) of their SECSs to access the sensitive services (e.g., provisioning service). We extend this mechanism to prevent other software components including malicious enclaves from falsifying *MMK* inside and outside the SGX-enabled processor. During the enclave initialization, LE checks if the initializing enclaves, except for ME, set their MIGRATIONs to true illegally by rejecting initialization requests from those enclaves. This check routine is possible because LE refers to the list of authorized enclaves and signs the initialization tokens (called EINITTOKEN) for the listed enclaves. Thus, only MEs can receive valid EINITTOKENs from LE among enclaves that attempt to set their MIGRATIONs to true. Without a valid EINITTOKEN, any enclave cannot be launched in the SGX-enabled processor (Costan and Devadas, 2016).

The SGX-enabled processor further checks if a caller enclave is ME by examining MIGRATION when the enclave invokes EPUTKEY. If MIGRATION of the enclave does not set to true, the SGX-enabled processor simply rejects the invocation of EPUTKEY. This two-step verification, which is enforced by LE and the SGX-enabled processor, convinces that only ME can execute EPUTKEY. As a result, *MMK* is only accessible by the designated enclaves (i.e., MEs) and the SGX-enabled processors.

4.2.3. *Diagram of key exchange with remote attestation*

Fig. 2 depicts the diagram of P_1. Recall that eMotion does not limit the underlying key exchange protocol if *MMK* is established between ME_S and ME_D based on remote attestation. Thus, the flows in Fig. 2 can vary slightly depending on used protocols. When VMM_S launches the migration, VMM_D also starts V with the exact parameters that VMM_S used. VMM_S initiates P_1 by establishing the network connection with VMM_D. Then, the VMMs request ME Hosts to execute MEs for operating the key exchange protocol. ME_D generates the key exchange message (msg1), and performs local attestation with its local QE. When local attestation succeeds, the QE produces QUOTE1 of msg1. ME_D sends msg1 and QUOTE1 to ME_S via VMM_D and VMM_S. ME_S verifies QUOTE1, and generates the key exchange message (msg2). Similarly, ME_S performs local attestation with its local QE, and receives QUOTE2 from the QE. ME_S sends msg2 and QUOTE2 to ME_D via VMM_S and VMM_D. Then, ME_S generates *MMK* using keying materials included in msg1 and msg2. ME_D verifies QUOTE2, and also generates *MMK*. Finally, MEs execute EPUTKEY to store the established *MMK* to MKR.

4.3. *SGX extension for secure eviction and loading for migration*

4.3.1. *Privileged instructions for migrating enclaves*

eMotion supports ESE (Enclave Secure Eviction) and ESL (Enclave Secure Loading) for VMM_S and VMM_D to evict and load the entire enclave pages securely. These instructions are ENCLS instructions, privileged instructions, that extend the EPC paging instructions (EWB, ELDU/B). Migration Key (*MK*) and Initialization Vector (*IV*) are derived from *MMK* inside the SGX-enabled processor during the initial execution of ESE and ESL. ESE and ESL utilize NIST SP 800–108 (Chen, 2009) as a key derivation function that the SGX implementation supports (Costan and Devadas, 2016). Suppose that $\mathtt{KDF}(\cdot)$ is the key derivation function that ESE and ESL use. Then, *MK* and *IV* are derived using Eq. (1).

$$\begin{aligned} MK &= \mathtt{KDF}(MMK, C_{MK}) \\ IV &= \mathtt{KDF}(MMK, C_{IV}) \end{aligned} \tag{1}$$

where C_{MK} and C_{IV} are constant string values.

ESE encrypts and integrity protects the migrated enclave pages using *MK* and *IV*. For anti-replay, *IV* increases by one for each EPC page, but *MK* does not change until the enclave migration completes. Therefore, the protection using *MK* and *IV* can preserve CIA of the migrated enclave pages. Because the SGX-enabled processors perform the derivation and the protection, it is impossible for other entities such as the VMMs to notice and falsify the migrated enclave pages.

This derivation can utilize VM identifiers to generate different *MK*s and *IV*s from each guest VM when VMM_S migrates multiple SGX-enabled VMs simultaneously. That is, ESE and ESL can derive multiple *MK*s and *IV*s for guest VMs by passing VM identifiers as input parameters to the underlying key derivation function. Suppose that VM_i is an VM identifier where $i = 1, 2, 3, \cdots$. Then, *MK* and *IV* for VM_i are derived using Eq. (2).

$$\begin{aligned} MK_i &= \mathtt{KDF}(MMK, C_{MK}, VM_i) \\ IV_i &= \mathtt{KDF}(MMK, C_{IV}, VM_i) \end{aligned} \tag{2}$$

where MK_i and IV_i are *MK* and *IV* for VM_i.

To reuse the SGX implementation, ESE and ESL execute routines similar to the EPC paging instructions. The cryptographic algorithm used by ESE and ESL is AES-GCM (Dworkin, 2007), which is used by the EPC paging instructions and supports both confidentiality and integrity. These instructions also take an input as the unit of a single EPC page like other SGX instructions.

VMM_S executes ESE to evict enclave pages of the running *E* from its PRM to the untrusted memory. ESE evicts the entire enclave pages, including PT_SECS, PT_TCS, and PT_REG. This instruction encrypts and integrity protects the enclave pages using *MK* and *IV*. VMM_D executes ESL to load the evicted enclave pages from the untrusted memory to its PRM. ESL loads the evicted enclave pages into the PRM, where PT_SECS, PT_TCS, and PT_REG reside. This instruction decrypts and integrity checks the evicted enclave pages using *MK* and *IV*.

Fig. 3 depicts the flow charts of ESE and ESL. For the sake of simplicity, the flow charts omit routines used to check the memory alignment.

ESE (Fig. 3a) works as follows:

1. Checks if the evicting page locates in EPC (if not, page fault exception (#PF) is raised).
2. Allocates the output addresses for the evicted EPC page and PCMD.
3. Searches EPCM to retrieve the metadata of the EPC page.
4. Searches the associated SECS if the EPC page type is PT_REG or PT_TCS.
5. Sets a temporary MAC header using the metadata in the searched EPCM.
6. Encrypts and integrity protects the EPC page.
7. Sets PCMD to complete the page information using the metadata in the searched EPCM.

ESL (Fig. 3b) works as follows:

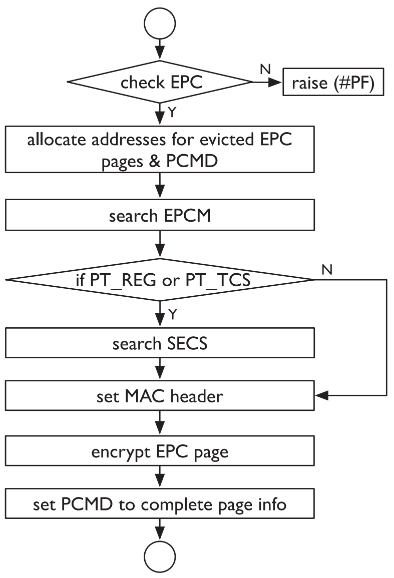
1. Checks if the loading page locates in EPC (if not, the page fault exception (#PF) is raised).
2. Allocates the input addresses of the evicted EPC page and PCMD.
3. Searches EPCM to retrieve the metadata of the loading EPC page.
4. Sets a temporary MAC header using the metadata in the searched EPCM.
5. Decrypts the EPC page.
6. Compares the computed MAC with the received one.
7. Sets EPCM using the decrypted metadata in the temporary MAC header.

4.3.2. *Diagram of secure eviction and loading for migration*

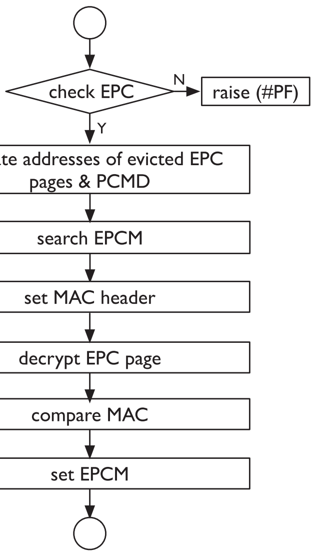
During executing *P* for *V*, VMM_S starts P_2 when VMM_S encounters the memory pages of *E* in its managed page table. The initial executions of ESE and ESL use *MMK* to derive *MK* and *IV*, which are utilized to evict and load the entire enclave pages securely. VMM_S executes ESE to evict the enclave pages of *E* using *MK* and *IV* from its PRM to the untrusted memory. Then, VMM_S transfers the evicted enclave pages to VMM_D. Once VMM_S transfers the evicted enclave pages, VMM_D executes ESL to load them using *MK* and *IV* from the untrusted memory to its PRM. Fig. 4 depicts the diagram of P_2.

During P_2, VMM_S checks if each enclave page alters by tracking the accessed and dirty flags of the enclave pages. The VMM utilizes the Extended Page Table (EPT) to manage the VM's address space that includes the enclave pages. Thus, VMM_S can notice the accessed and dirtied EPC pages by scanning the EPT. When detecting the updated enclave pages, VMM_S executes ESE against the updated enclave pages and retransmits the output to VMM_D.

VMM_S can also transfer the swapped enclave pages to VMM_D during VM migration. Because VMM_S swapped out the enclave pages due to the lack of its PRM, VMM_S can be aware of the swapped enclave pages. However, VMM_S cannot notice the swapped enclave pages if *V* swaps out the enclave pages by itself. This mismatch between VMM_S and *V* can be addressed if VMM_S emulates SGX instructions for *V* (Chakrabarti et al., 2017). To migrate the swapped enclave pages, VMM_S executes the EPC paging instructions to load them to its PRM again and continues to execute ESE for evicting the enclave pages.



(a) Flow chart of ESE



(b) Flow chart of ESL

Fig. 3 – Flow charts of ESE and ESL.

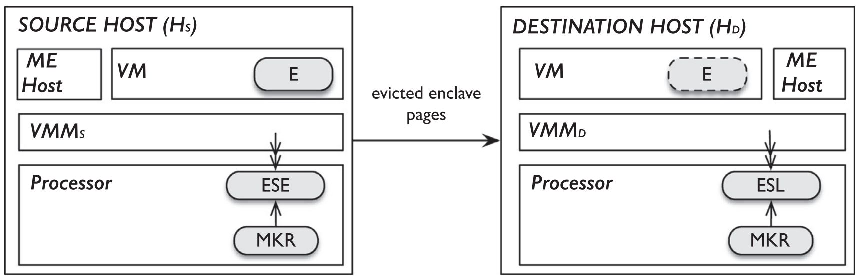


Fig. 4 – Diagram of secure eviction and loading for migration; register read/write ($\rightarrow$), and instruction execution ($\twoheadrightarrow$).

4.4. *eMotion architecture*

Fig. 5 depicts an architecture to show the practical deployment of eMotion. We assume that VMM_S in H_S migrates V along with E to VMM_D in H_D.

P_1 establishes MMK for enclave migration between H_S and H_D. When starting P to migrate V along with E, VMM_S triggers ME Host, a daemon running in the host, to execute ME_S. ME_S proves its authenticity to QE based on local attestation and to ME_D based on remote attestation. In consequence of remote attestation, ME_S and ME_D establish MMK and execute EPUTKEY to store MMK into MKRs of the SGX-enabled processors.

In P_2, VMM_S executes ESE to evict the enclave pages of E to the untrusted memory, and VMM_D executes ESL loads them to its PRM. Because the VMMs manage the memory mappings of the VM and its enclaves, the VMMs can pass the physical addresses of the enclave pages (in the PRM) and the evicted enclave pages (in the untrusted memory) to ESE and ESL, respectively. After loading the enclave pages, VMM_D activates V along with the enclaves including E using S received from VMM_S. Because VMM_S transfers VM's whole memory pages including the entire enclave pages and VM state to VMM_D, VMM_D can restore the execution of V and E.

5. Implementation

We have implemented a prototype of eMotion on top of OpenSGX (Jain et al., 2016) in a Dell Inspiron-13-7359 (Intel Core i5-6200 2.30GHz quad core CPU, 8GB RAM) machine running Ubuntu 14.04 LTS (64-bit). Using this open source SGX emulator, we add additional instructions and migration support to demonstrate the operations of eMotion.

5.1. *OpenSGX*

OpenSGX is an open source SGX emulator that emulates the SGX instructions and provides operating components. This emulator is implemented on top of QEMU's user-mode emulation. OpenSGX extends the CPU state of QEMU by adding CREGS data structure. CREGS maintains registers about the enclave context and the current instruction pointer. This data structure controls a program's next executing point when the enclave enters and exits. OpenSGX utilizes the QEMU

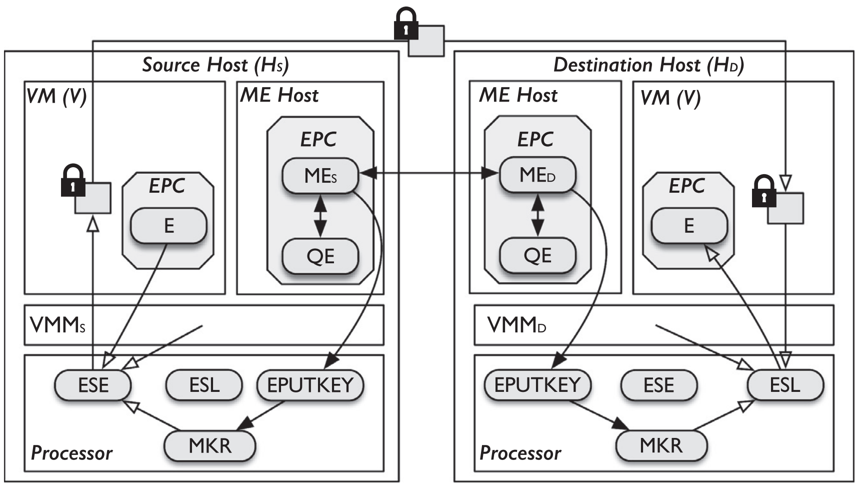


Fig. 5 – An architecture of eMotion; key exchange with remote attestation (—►), and secure eviction and loading for migration (—▷).

helper routine and adds `helper_sgx_encls(u)` for emulating ENCLU/ENCLS instructions. When ENCLU/ENCLS instructions are invoked, the helper functions implemented in `helper_sgx_encls(u)` are called.

5.2. *eMotion on OpenSGX*

We implement three new SGX instructions (EPUTKEY, ESE, ESL), migration support (MEs, MKR and MIGRATION), and other SGX components (QEs) to OpenSGX. We implement MEs to operate a sample of P_1 based on the 1024-bit DH key exchange protocol (Diffie and Hellman, 1976) to establish *MMK*. We also implement QEs to use a pre-defined RSA key pair for signing each key exchange message from MEs and verifying each `REPORT` from MEs. QE and ME utilize PolarSSL (PolarSSL Project) for local and remote attestation. We add EPUTKEY to `helper_sgx_enclu`, and insert MKR into the `CREGS` data structure of QEMU SGX. We add ESE and ESL to `helper_sgx_encls` and implement the routine to derive *MK* and *IV* when these privileged instructions execute for the first time. To support CIA of the evicted enclave pages, ESE and ESL leverage OpenSSL 1.0.2d (OpenSSL Project) to encrypt and integrity protect the enclave pages based on AES-GCM (Dworkin, 2007).

We add a new OpenSGX application (hereafter, vmm) that acts as the VMMs (VMM_S and VMM_D) for executing P_M. After P_1 completes, vmm in each host calls the functions of the OS-level emulation wrappers for ESE and ESL. Once both hosts complete P_2, vmm in H_D attempts to re-enter the migrated enclave (*E*) and check if enclave migration has been completed.

5.3. *Implementation result*

In the current prototype, we add a total of 2,286 lines of code to OpenSGX and confirm the operations of eMotion. Fig. 6 shows the implementation result of the prototype. We describe the execution steps as follows:

1. ME_S requests ME_D to start P_1. Then, ME_D generates `msg1`, and requests the destination QE to generate `QUOTE1` (Fig. 6e).
2. When local attestation succeeds, ME_D responds with `QUOTE1` of `msg1` for remote attestation (Fig. 6f).
3. When receiving `msg1` and `QUOTE1`, ME_S verifies `QUOTE1`. If remote attestation succeeds, ME_S generates `msg2`, and requests the source QE to generate `QUOTE2` (Fig. 6b).
4. When local attestation succeeds, the source QE responds with `QUOTE2` of `msg2` for remote attestation (Fig. 6c).
5. When receiving `msg2` and `QUOTE2`, ME_D verifies `QUOTE2`. If remote attestation succeeds, ME_D generates *MMK* based on `msg1` and `msg2`. In the same way, ME_S generates *MMK* (Fig. 6b and e).
6. MEs cooperate with Es in both hosts for executing EPUTKEY to store *MMK* to MKR (Fig. 6a and d).
7. The vmm in H_S executes ESE to evict the entire enclave pages of *E* (Fig. 6a).
8. The vmm in H_D executes ESL to load the evicted enclave pages of *E*. Then, vmm again launches *E* to check if *E* is migrated successfully (6d).

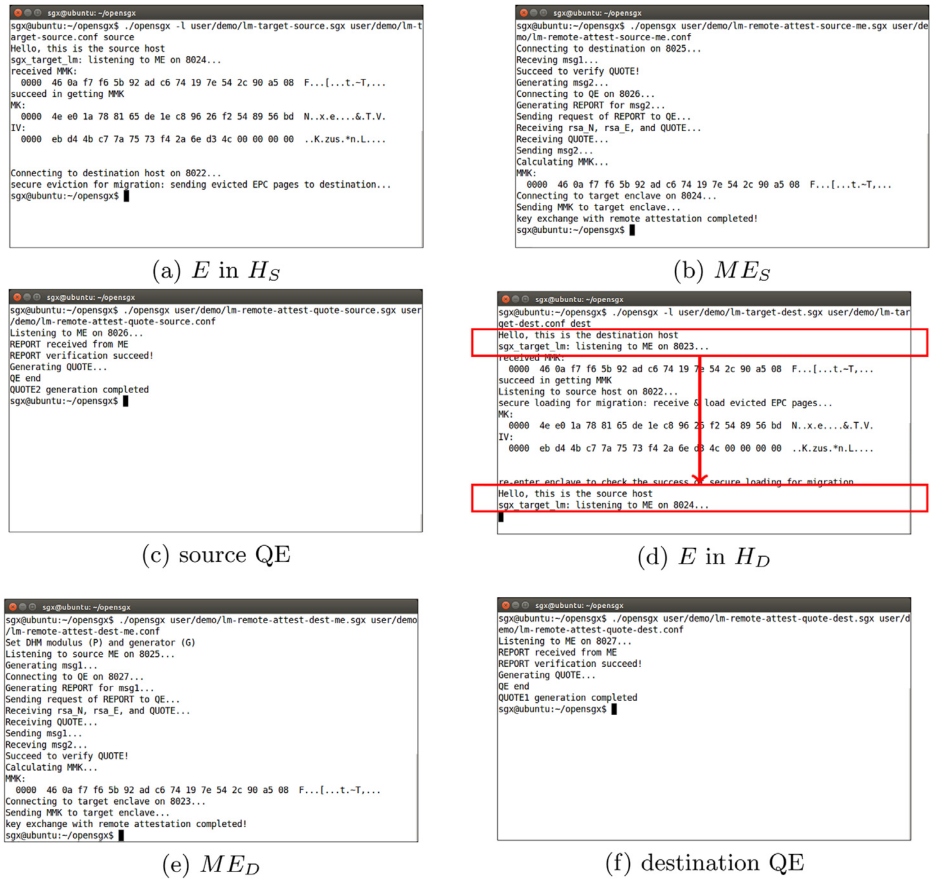
As indicated by the arrow in Fig. 6d, vmm migrates *E* using eMotion.

6. Evaluation

6.1. *Analysis*

We evaluate eMotion from the perspective of the goals defined in Section 3.

In P_1, ME_S and ME_D in both hosts perform remote attestation to convince that two legitimate enclaves establish *MMK* because local and remote attestation can vouch for this authentication. Additionally, it promises that two legitimate SGX-enabled processors execute this protocol because only genuine processors can perform remote attestation successfully. During P_2, ESE encrypts and integrity protects the en-



```
sgx@ubuntu: ~/opensgx
sgx@ubuntu:~/opensgx$ ./opensgx -l user/demo/lm-target-source.sgx user/demo/lm-t
arget-source.conf source
Hello, this is the source host
sgx_target_lm: listening to ME on 8024...
received MMK:
  0000  46 0a f7 f6 5b 92 ad c6 74 19 7e 54 2c 90 a5 08  F...[...t.~T,...
succeed in getting MMK
MK:
  0000  4e e0 1a 78 81 65 de 1e c8 96 26 f2 54 89 56 bd  N..x.e....&.T.V.
IV:
  0000  eb d4 4b c7 7a 75 73 f4 2a 6e d3 4c 00 00 00 00  ..K.zus.*n.L....


Connecting to destination host on 8022...
secure eviction for migration: sending evicted EPC pages to destination...
sgx@ubuntu:~/opensgx$ █
```

(a) E in H_S

```
sgx@ubuntu: ~/opensgx
sgx@ubuntu:~/opensgx$ ./opensgx user/demo/lm-remote-attest-source-me.sgx user/de
mo/lm-remote-attest-source-me.conf
Connecting to destination on 8025...
Receving msg1...
Succeed to verify QUOTE!
Generating msg2...
Connecting to QE on 8026...
Generating REPORT for msg2...
Sending request of REPORT to QE...
Receiving rsa_N, rsa_E, and QUOTE...
Receiving QUOTE...
Sending msg2...
Calculating MMK...
MMK:
  0000  46 0a f7 f6 5b 92 ad c6 74 19 7e 54 2c 90 a5 08  F...[...t.~T,...
Connecting to target enclave on 8024...
Sending MMK to target enclave...
key exchange with remote attestation completed!
sgx@ubuntu:~/opensgx$ █
```

(b) ME_S

```
sgx@ubuntu: ~/opensgx
sgx@ubuntu:~/opensgx$ ./opensgx user/demo/lm-remote-attest-quote-source.sgx user
/demo/lm-remote-attest-quote-source.conf
Listening to ME on 8026...
REPORT received from ME
REPORT verification succeed!
Generating QUOTE...
QE end
QUOTE2 generation completed
sgx@ubuntu:~/opensgx$ █
```

(c) source QE

```
sgx@ubuntu: ~/opensgx
sgx@ubuntu:~/opensgx$ ./opensgx -l user/demo/lm-target-dest.sgx user/demo/lm-tar
get-dest.conf dest
Hello, this is the destination host
sgx_target_lm: listening to ME on 8023...
received MMK:
  0000  46 0a f7 f6 5b 92 ad c6 74 19 7e 54 2c 90 a5 08  F...[...t.~T,...
succeed in getting MMK
Listening to source host on 8022...
secure loading for migration: receive & load evicted EPC pages...
MK:
  0000  4e e0 1a 78 81 65 de 1e c8 96 26 f2 54 89 56 bd  N..x.e....&.T.V.
IV:
  0000  eb d4 4b c7 7a 75 73 f4 2a 6e d3 4c 00 00 00 00  ..K.zus.*n.L....


re-enter enclave to check the success of secure loading for migration
Hello, this is the source host
sgx_target_lm: listening to ME on 8024...
█
```

(d) E in H_D

```
sgx@ubuntu: ~/opensgx
sgx@ubuntu:~/opensgx$ ./opensgx user/demo/lm-remote-attest-dest-me.sgx user/demo
/lm-remote-attest-dest-me.conf
Set DHM modulus (P) and generator (G)
Listening to source ME on 8025...
Generating msg1...
Connecting to QE on 8027...
Generating REPORT for msg1...
Sending request of REPORT to QE...
Receiving rsa_N, rsa_E, and QUOTE...
Receiving QUOTE...
Sending msg1...
Receving msg2...
Succeed to verify QUOTE!
Calculating MMK...
MMK:
  0000  46 0a f7 f6 5b 92 ad c6 74 19 7e 54 2c 90 a5 08  F...[...t.~T,...
Connecting to target enclave on 8023...
Sending MMK to target enclave...
key exchange with remote attestation completed!
sgx@ubuntu:~/opensgx$ █
```

(e) ME_D

```
sgx@ubuntu: ~/opensgx
sgx@ubuntu:~/opensgx$ ./opensgx user/demo/lm-remote-attest-quote-dest.sgx user/d
emo/lm-remote-attest-quote-dest.conf
Listening to ME on 8027...
REPORT received from ME
REPORT verification succeed!
Generating QUOTE...
QE end
QUOTE1 generation completed
sgx@ubuntu:~/opensgx$ █
```

(f) destination QE

Fig. 6 – Implementation result of eMotion in OpenSGX.

clave pages and ESL integrity checks and decrypts the evicted enclave pages using *MK* and *IV*, which are derived from *MMK*. *IV* increases by one for each EPC page to guarantee anti-replay. Thus, the migrated enclave pages can guarantee CIA during enclave migration. During P_1 and P_2, no additional trusted server involves, but rather two participating hosts establish *MMK* directly to provide the end-to-end protection on the migrated enclave pages. *Adv*. cannot acquire *MMK* because it cannot access the EPC pages directly, which is protected by the SGX-enabled processor, and the access on *MMK* is restricted only to MEs and the SGX-enabled processors. *Adv*. cannot read the migrated enclave pages in plain-text because the SGX-enabled processor encrypts the migrated enclave pages. Moreover, *Adv*. cannot violate the security properties of the migrated enclave pages during enclave migration because ESE and ESL guarantee CIA of the migrated enclave pages (**G1**). eMotion stores *MMK* in MKR of the SGX-enabled processor and restricts the execution of EPUTKEY only to MEs. Thus, no other software including the VMs and the VMMs can use or change *MMK* illegally. This restriction on EPUTKEY is not avoidable because LE prevents other enclaves from setting MIGRATION during the enclave initialization, and the SGX-enabled processor checks if the caller sets MIGRATION when EPUTKEY is invoked. *Adv*. cannot hijack *MMK* established by MEs as well because they are infeasible to access directly the EPC pages and MKR of the SGX-enabled processor where *MMK* resides (**G2**).

We add new SGX instructions (ESE and ESL) to evict and load the entire enclave pages including ones that cannot be evicted and loaded by the existing EPC paging instructions. During P_2, VMM_S can evict the entire enclave pages of the running *E* to the untrusted memory using ESE and VMM_D can load the evicted enclave pages to its PRM using ESL. To migrate the running *E*, VMM_S transfers the update enclave pages continually and transfers its running state S to VMM_D at the end of P_M. Because of this operation, VMM_D can restore the memory mappings for *E* and activate the execution of *E*. Thus, using newly added SGX instructions, two VMMs can migrate the running *E* (**G3**).

6.2. *Performance*

We measure the overhead of eMotion to estimate the impact on the migration time and migration downtime of SGX-enabled VMs. We use the prototype based on OpenSGX as mentioned in Section 5. Though this performance evaluation is not measured in the actual SGX-enabled machine, we expect that these results can help others understand and estimate the overheads caused by managed enclave migration.

6.2.1. *Overhead in key exchange with remote attestation*

Table 1 shows the overhead caused by key exchange with remote attestation in ME and QE in terms of the number of in-

Table 1 – The number of instructions in ME and QE during key exchange with remote attestation.

	ME	QE
SGX instructions	110	39
Normal instructions	144M	14M

Table 2 – Elapsed time for secure eviction and loading for migration.

	VMM_S	VMM_D
Elapsed time (ms)	6.69 ms	4.31 ms

structions. We refer to the model used in Kim et al. (2015) to calculate CPU cycles consumed by ME and QE for operating P_1. Therefore, we assume that each SGX instruction consumes 10K CPU cycles (Baumann et al., 2015), and uses 1.8 CPU cycles for each normal instruction (Kim et al., 2015). ME consumes 259M cycles to perform 1024-bit DH key exchange protocol and derive *MMK* together with local and remote attestation. QE consumes 25M cycles to generate and verify QUOTE. Note that P_1 occurs only once before the actual migration starts. Thus, this overhead is minimal and does not affect the migration downtime.

6.2.2. *Overhead in secure eviction and loading for migration*

We also measure the overhead caused by secure eviction and loading for migration in VMM_S and VMM_D. For this, we implement an sample enclave, which occupies 616 EPC pages including PT_SECS, PT_TCS, and PT_REG. Recall that VMM_S and VMM_D execute ESE and ESL for each memory page in the unit of a single EPC page. Obviously, the number of instructions that VMM_S and VMM_D execute in P_2 changes according as the number of EPC pages that consists of *E* increases.

Because the overhead in P_2 influences the migration downtime directly, we measure the elapsed time for P_2. Table 2 reports that the elapsed time for P_2 in VMM_S is about 6.69 *ms* and the one for P_1 in VMM_D is about 4.31 *ms*. As shown in Fig. 3, the additional routines used to check the condition and search SECS in ESE cause this gap between two measured times.

Using the measured time, we can further estimate the elapsed time for a single ESE (10.9 us) and ESL (7.0 us). Besides the sample enclave, we calculate the elapsed time to migrate Tor enclaves used as a case study for OpenSGX. The Tor enclaves include Directory node (472 EPC pages) and Exit node (475 EPC pages) as mentioned in Jain et al. (2016). Fig. 7 shows the elapsed time for secure eviction and loading for migration on the enclaves; our sample enclave, Directory node, and Exit node. This estimation can help cloud tenants to profile the impacts of their SGX-enabled VMs during live migration.

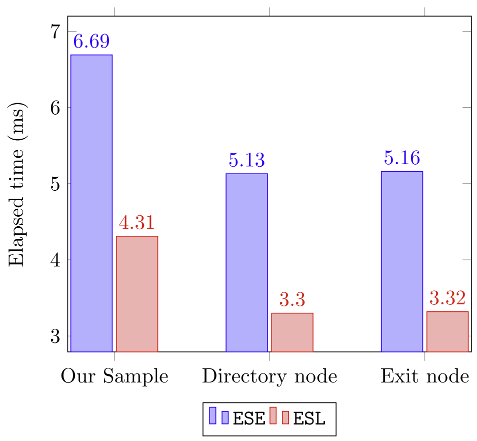


Fig. 7 – Elapsed time for secure eviction and loading for migration on enclaves; Directory node and Exit node are Tor enclaves in Jain et al. (2016).

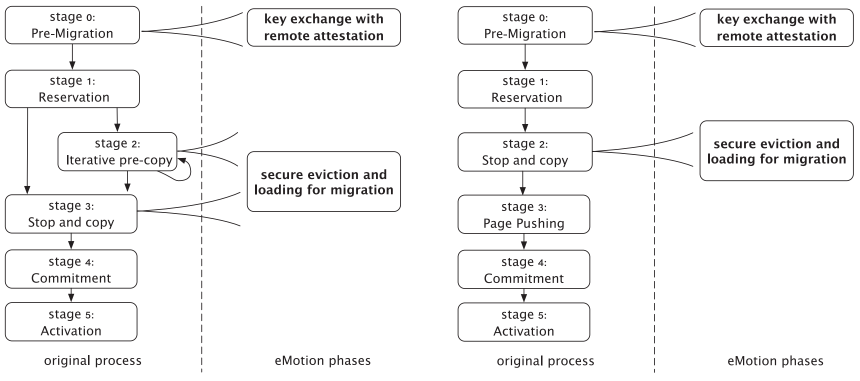
7. Discussion

7.1. *Possible deployments*

The existing live migration of VMs (Clark et al., 2005; Hines and Gopalan, 2009) can use eMotion by adding two phases, as depicted in Fig. 8. Key exchange with remote attestation occurs during the pre-migration stage to establish the migration master key between two participating hosts before the actual live migration of the SGX-enabled VM begins. During the iterative pre-copy and/or the stop and copy stages, secure eviction and loading for migration operates using the established migration master key when the VMM encounters the EPC pages. Similarly, other live migration protocols (Sahni and Varma, 2012) can add eMotion to support live migration of SGX-enabled VMs.

7.2. *Limitations*

This paper mainly focuses on the extensions of SGX implementation to enable managed enclave migration. The prototype based on OpenSGX confirms the operations of eMotion. However, we cannot confirm the operations of the eMotion-enabled VMMs because OpenSGX, which uses the user-mode QEMU emulation, does not run on top of the VMM. eMotion does not consider that the enclave has the sealed data, which is encrypted by a unique key inside the SGX-enabled processor. The VMM cannot notice the sealed data because the enclave performs the sealing operation by itself. Thus, migrating the sealed data of the enclave is another challenging problem. eMotion can extend to cope with the attacks that the active adversaries can perform by combining with the existing security mechanisms. The active adversaries that subvert the VMMs can incur the state inconsistency in the migrated enclave pages by preventing the VMMs from tracking the updated enclave pages. The existing VMM attestation mechanisms (Greene, 2012; TCG, 2012) can launch before enclave migration to verify if the genuine VMM launched and is running on the source host. Moreover, users can utilize the



(a) Pre-copy approach [21] (b) Post-copy approach [22]

Fig. 8 – Possible deployments of eMotion to existing live migration of VMs.

existing detection mechanism (Brandenburger et al., 2017) to detect the rollback and forking attacks that the active adversaries can perform.

7.3. *Future work*

We will study about migrating the sealed data of the enclave securely. We also will research other SGX emulators like S-OpenSGX (Choi et al., 2017) that run on top of the VMM so that we realize the eMotion-enabled VMM. Then, our prototype will extend to confirm the operations of the eMotion-enabled VMM.

8. Related work

8.1. *Secure live migration of SGX enclaves*

Enclave migration is one of the technically challenging issues for introducing SGX into cloud computing, and thus few peer-reviewed papers on enclave migration can be found (Gu et al., 2017; Park et al., 2016). Instead, we refer to Intel's patents (Rozas et al., 2017; 2018) as supplementary references.

Our previous work (Park et al., 2016) identified problems in live migration of SGX-enabled VMs and presented a conceptual scheme to address the problems without the actual implementation. Gu et al. (2017) presented a secure enclave migration in a self-migration manner. They first introduced an attack that causes data inconsistency and control inconsistency when the self-migration manner of the enclave occurs and proposed two-phase checkpointing to deal with the attack. Only the control thread running in the migrated enclave can access the encryption key and the integrity key for protecting the migrated enclave pages. This approach necessitates the enclave to use an additional library that supports enclave migration, and thus it is impossible to migrate the enclave without the specific library. Because this work is on the basis of the self-migration manner, the authors presented only conceptual design suggestions for new SGX instructions.

Intel presented two patents (Rozas et al., 2017; 2018) to enable live migration of SGX-enabled VMs in the managed migration manner. Intel defines SGX domain control structure (SDCS) that stores the migration capable keys, which are generated by the controlling enclave. The source host transmits SDCS to the destination host via the trusted server, and this SDCS protects the enclave pages for secure enclave migration. Intel also presents the instructions for migrating the enclave using SDCS. Currently, the practical implementation of these patents is not realized yet. Moreover, the transportation of the migration capable keys is still conceptual and needs the additional trusted server. Because the trusted server mediates the transportation of the migration capable keys, it is difficult to ensure that only the participating hosts can access the migration capable keys. Accordingly, the end-to-end protection on the migrated enclave pages cannot be guaranteed.

Table 3 shows this comparison between eMotion and the existing migration schemes for SGX enclaves against the goals defined in Section 3.3. However, we cannot perform the experimental comparison among eMotion and other approaches because the Intel's patents(Rozas et al., 2017; 2018) are not implemented yet and the source code of Gu et al. (2017) is not available.

8.2. *SGX in cloud computing and virtualization*

SGX has been leveraged to isolate and protect code and data in cloud computing and virtualization (Baumann et al., 2015; Schuster et al., 2015; Dinh et al., 2015; Hunt et al., 2016;

Table 3 – Comparison with the existing migration schemes for SGX enclaves.

	eMotion	Gu et al. (2017)	Intel's patents (Rozas et al., 2017; 2018)
G1	◯	◯	×
G2	◯	◯	×
G3	Managed (◯)	Self (×)	Managed (◯)

Arnautov et al., 2016; Bhardwaj et al., 2016). Baumann et al. (2015) provides shielded execution of unmodified applications. VC3 (Schuster et al., 2015) and M^{2}R (Dinh et al., 2015) present secure and privacy-preserving MapReduce computations, respectively. Hunt et al. (2016) is a distributed sandbox to protect sandbox instances, and SCONE (Arnautov et al., 2016) is a secure container mechanism for Docker. AirBox (Bhardwaj et al., 2016) supports fast, scalable and secure edge functions for device-cloud interactions. Brandenburger et al. (2017) introduced Lightweight Collective Memory (LCM) to detect rollback and forking attacks by using a distributed protocol for maintaining consistency information by clients. LCM addresses the important trust issues in cloud computing, but its migration mechanism needs an enclave to stop its processing and to supplement with additional implementation for enclave migration. These SGX-based mechanisms leverage the hardware-assisted isolation and protection of applications' code and data, so the inherent security concerns raised in the fields have been addressed. However, without the consideration of enclave migration, these approaches can suffer from the management problems such as fault management, load balancing, system maintenance, etc. Hence, eMotion can alleviate these concerns without the loss of security guaranteed by SGX.

Recently, Intel extends the SGX support to the VMM-based virtualization (Intel Corporation, 2016a; Chakrabarti et al., 2017). Intel released the patches for SGX virtualization that enable KVM or Xen guest VMs to run enclaves (Intel Corporation, 2016a). Intel also presented the SGX Oversubscription Extensions (Chakrabarti et al., 2017) that overcome the difficulties in virtualizing SGX memory using the existing EPC paging swapping.

9. Conclusion

In this paper, we propose eMotion, an SGX extension for migrating enclaves, that adds additional instructions and migration support to the SGX architecture for enabling the secure managed migration of running enclaves. eMotion supplements the current SGX implementation with three SGX instructions, one register, one SECS attribute and one AE for migrating the running enclave. Using eMotion, the participating hosts directly establish the migration master key used in enclave migration and the VMMs in the hosts migrate running enclaves using this established key without the loss of the security properties guaranteed by SGX. eMotion restricts the access on the migration master key only to the designated AEs (MEs) and the SGX-enabled processors. We implement a prototype on top of OpenSGX, an open source SGX emulator, to demonstrate the operations of eMotion and to estimate the impact of eMotion on the migration time and the migration downtime. We hope that Intel refers to eMotion for realizing managed enclave migration in the actual SGX-enabled processor and the SGX framework, and cloud tenants use the evaluation result to estimate the impact of eMotion on their SGX-enabled VMs during live migration. As future work, we will study about migrating the sealed data of enclave, and extend our prototype to realize the eMotion-enabled VMM.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Anati I, Gueron S, Johnson S, Scarlata V. Innovative Technology for CPU Based Attestation and Sealing. Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, 2013.

Arnautov S, Trach B, Gregor F, Knauth T, Martin A, Priebe C, Lind J, Muthukumaran D, OKeeffe D, Stillwell ML, et al. SCONE: Secure linux containers with Intel SGX, 16; 2016. p. 689–703.

Baumann A, Peinado M, Hunt G. Shielding Applications from an Untrusted Cloud with Haven. ACM Trans. Comput. Syst. (TOCS) 2015;33(3) 8:1-8:26.

Bhardwaj K, Shih M-W, Agarwal P, Gavrilovska A, Kim T, Schwan K. Fast, scalable and secure onloading of edge functions using AirBox. In: Proceedings of the IEEE/ACM symposium on edge computing (SEC). IEEE; 2016. p. 14–27.

Brandenburger M, Cachin C, Lorenz M, Kapitza R. Rollback and Forking Detection for Trusted Execution Environments using Lightweight Collective Memory. In: Proceedings of the 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE; 2017. p. 157–68.

Chakrabarti S, Leslie-Hurd R, Vij M, McKeen F, Rozas C, Caspi D, Alexandrovich I, Anati I. Intel Software Guard Extensions (Intel SGX) Architecture for Oversubscription of Secure Memory in a Virtualized Environment. Proceedings of the Hardware and Architectural Support for Security and Privacy. ACM, 2017.

Chen L. Rec ommendation for key derivation using pseudorandom functions, 2009, (https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-108.pdf). Last accessed 2018-05-12.

Choi C, Kwak N, Jang J, Jang D, Oh K, Kwag K, Kang BB. S-OpenSGX: a system-level platform for exploring SGX enclave-based computing. Comput. Secur. 2017;70:290–306.

Clark C, Fraser K, Hand S, Hansen JG, Jul E, Limpach C, Pratt I, Warfield A. Live migration of virtual machines. In: Proceedings of the 2nd conference on symposium on networked systems design & implementation-Volume 2. USENIX Association; 2005. p. 273–86.

Costan V, Devadas S. Intel SGX Explained, 2016, (Cryptology ePrint Archive, Report 2016/086). https://eprint.iacr.org/2016/086, last accessed 2018-05-04.

CVE-2015-3340: Information leak through XEN_DOMCTL_gettscinfo, 2018 http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3340 last accessed 2018-04-29.

Diffie W, Hellman M. New Directions in Cryptography. IEEE Trans. Inform. Theory 1976;22(6):644–54.

Dinh TTA, Saxena P, Chang E-C, Ooi BC, Zhang C. M^{2}R: enabling stronger privacy in mapreduce computation. In: Proceedings of the USENIX Security Symposium; 2015. p. 447–62.

Dworkin MJ, Recommendation for block cipher modes of operation: galois/counter mode (GCM) and GMAC, 2007, (https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=51288). Last accessed 2018-04-29.

Greene J, Intel trusted execution technology - hardware-based technology for enhancing server platform security, 2012, (https://www.intel.de/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf). Last accessed 2018-04-29.

Gu J, Hua Z, Xia Y, Chen H, Zang B, Guan H, Li J. Secure Live Migration of SGX Enclaves on Untrusted Cloud. In: Proceedings of the 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE; 2017. p. 225–36.

Hines MR, Gopalan K. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. ACM; 2009. p. 51–60.

Hunt T, Zhu Z, Xu Y, Peter S, Witchel E. Ryoan: a distributed sandbox for untrusted computation on secret data. In: Proceedings of the 12th USENIX symposium on operating systems design and implementation (OSDI 16). USENIX Association; 2016. p. 533–49.

Intel Corporation, SGX Virtualization, 2016a, (https://www.01.org/intel-software-guard-extensions/sgx-virtualization a). Last accessed 2018-04-29.

Intel Corporation, Intel Software Guard Extensions Developer Guide, 2016b, (https://www.01.org/sites/default/files/documentation/intel_sgx_developer_guide_pdf.pdf b). Last accessed 2018-04-29.

Intel Corporation, Intel software guard extensions programming reference, Intel Corporation2014, (https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf). Last accessed 2018-04-29.

Intel Corporation, 2018Attestation service for intel software guard extensions (Intel SGX): API documentation, 2018, Intel Corporation (https://www.software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf) Last accessed 2018-05-22

Jain P, Desai S, Kim S, Shih M-W, Lee J, Choi C, Shin Y, Kim T, Kang BB, Han D. OpenSGX: an open platform for SGX research. Proceedings of the Network and Distributed System Security Symposium, 2016.

Kim S, Shin Y, Ha J, Kim T, Han D. A first step towards leveraging commodity trusted execution environments for network applications. Proceedings of the 14th ACM Workshop on Hot Topics in Networks. ACM, 2015.

OpenSSL Project, 2018 (https://www.openssl.org/ b). Last accessed 2018-04-29.

Park J, Park S, Oh J, Won J. Toward live migration of SGX-enabled virtual machines. In: Proceedings of the 2016 IEEE World Congress on Services (SERVICES). IEEE; 2016. p. 111–12.

PolarSSL Project, 2018 (https://www.polarssl.org/ a). Last accessed 2018-04-29.

Rozas CV, Vij M, Leslie-Hurd RM, Zmudzinski KC, Chakrabarti S, McKeen FX, Scarlata VR, Johnson SP, Alexandrovich I, Neiger G, et al., Processors, methods, systems, and instructions to support live migration of protected containers, 2017, US Patent 9,710,401.

Rozas CV, Vij M, Leslie-Hurd RM, Zmudzinski KC, Chakrabarti S, McKeen FX, Scarlata VR, Johnson SP, Alexandrovich I, Platform migration of secure enclaves, 2018, US Patent 9,942,035.

Sahni S, Varma V. A hybrid approach to live migration of virtual machines. In: 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM). IEEE; 2012. p. 1–5.

Schuster F, Costa M, Fournet C, Gkantsidis C, Peinado M, Mainar-Ruiz G, Russinovich M. VC3: trustworthy data analytics in the cloud using SGX. In: Proceedings of the 2015 IEEE Symposium on Security and Privacy. IEEE; 2015. p. 38–54.

TCG, TCG PC client specific implementation specification for conventional BIOS, 2012, (https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_PCClientImplementation_1-21_1_00.pdf). Last accessed 2018-04-29.

Jaemin Park received the B.S. degree from Handong Global University in 2004 and the M.S. degree from Korea Advanced Institute of Science and Technology (KAIST), South Korea in 2006. He is a senior researcher at the Affiliated Institute of Electronics and Telecommunications Research Institute (ETRI) and a Ph.D. candidate in Graduate School of Information Security, KAIST. His research interests include cloud computing, system security, and trusted execution environments especially in Intel SGX.

Sungjin Park received the B.S. degree from Inha University in 2002, the M.S. degree from POSTECH in 2005, and the Ph.D. degree in Computer Science, Korea Advanced Institute of Science and Technology (KAIST), South Korea in 2017. He is a senior researcher at the Affiliated Institute of Electronics and Telecommunications Research Institute (ETRI). His research interests include cloud computing and system security.

Brent Byunghoon Kang is currently an associate professor at the Graduate School of Information Security at Korea Advanced Institute of Science and Technology (KAIST). Before KAIST, he has been with George Mason University as an associate professor. Dr. Kang received his Ph.D. in Computer Science from the University of California at Berkeley, and M.S. from the University of Maryland at College Park, and B.S. from Seoul National University. He has been working on systems security area including botnet defense, OS kernel integrity monitors, trusted execution environment, and hardware-assisted security. He is currently a member of the IEEE, the USENIX and the ACM.

Kwangjo Kim received the B.Sc. and M.Sc. degrees in Electronic Engineering from Yonsei University, Seoul, South Korea, in 1980 and 1983, respectively, and the Ph.D. degree from the Division of Electrical and Computer Engineering, Yokohama National University, Japan, in 1991. He was a visiting professor with MIT and UCSD, USA, in 2005, and the Khalifa University of Science, Technology and Research (KUSTAR), Abu Dhabi, UAE, in 2012, and an Education Specialist with the Bandung Institute of Technology (ITB), Bandung, Indonesia, in 2013. He is currently a full professor at the Graduate School of Information Security, School of Computing, Korea Advanced Institute of Science and Technology (KAIST), South Korea, the Korean representative to IFIP TC-11, and the honorable president of the Korea Institute of Information Security and Cryptography (KIISC). His current research interests include the theory of cryptology and information security and their practices. Prof. Kim had served as a board member of the IACR from 2000 to 2004, the Chairperson of Asiacrypt Steering Committee from 2005 to 2008, and the President of KIISC in 2009. He is a fellow of the IACR and a member of the IEEE, the ACM, the IACR, and the IEICE.