

박 사 학 위 논 문  
Ph.D. Dissertation



SGX를 이용한 클라우드 가상사설망의 보안 강화  
및 Enclave 마이그레이션 기법 연구

Security-Enhanced Cloud VPN with SGX and Enclave Migration

2019

박 재 민 (朴宰民 Park, Jaemin)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

SGX를 이용한 클라우드 가상사설망의 보안 강화  
및 Enclave 마이그레이션 기법 연구

2019

박 재 민

한 국 과 학 기 술 원

전산학부 (정보보호대학원)

# SGX를 이용한 클라우드 가상사설망의 보안 강화 및 Enclave 마이그레이션 기법 연구

박 재 민

위 논문은 한국과학기술원 박사학위논문으로  
학위논문 심사위원회의 심사를 통과하였음

2019년 5월 20일

심사위원장 김 광 조 (인)

심 사 위 원 윤 현 수 (인)

심 사 위 원 강 병 훈 (인)

심 사 위 원 한 동 수 (인)

심 사 위 원 신 욱 (인)

# Security-Enhanced Cloud VPN with SGX and Enclave Migration



Jaemin Park

Advisor: Kwangjo Kim

A dissertation submitted to the faculty of  
Korea Advanced Institute of Science and Technology in  
partial fulfillment of the requirements for the degree of  
Doctor of Philosophy in Computer Science (Information Security)

Daejeon, Korea  
May 20, 2019

Approved by

---

Kwangjo Kim  
Professor of Computing

The study was conducted in accordance with Code of Research Ethics<sup>1</sup>.

---

<sup>1</sup> Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.



DIS  
20155502

박재민. SGX를 이용한 클라우드 가상사설망의 보안 강화 및 Enclave 마이그레이션 기법 연구. 전산학부 (정보보호대학원) . 2019년. 67+v 쪽.  
지도교수: 김광조. (영문 논문)

Jaemin Park. Security-Enhanced Cloud VPN with SGX and Enclave Migration. School of Computing (Graduate School of Information Security) . 2019. 67+v pages. Advisor: Kwangjo Kim. (Text in English)

## 초 록

KAIST

이 논문은 SGX를 이용한 클라우드 가상사설망의 보안 강화 및 enclave 마이그레이션 기법을 다루었다. 클라우드 가상사설망은 온-프레미스 네트워크와 가상 사설 클라우드 네트워크 간의 통신을 보호하는 필수적인 네트워크 인프라이다. 하지만 반신뢰 클라우드 환경에서의 하이퍼바이저 취약점, 악의적인 클라우드 관리 기능 등으로 야기되는 정보 유출로 인해 클라우드 가상사설망의 프라이버시 문제가 발생할 수 있다. 가상사설망 서비스를 공유 환경에서 제공하는 낮은 수준의 격리를 활용하는 기존 연구들은 반신뢰 클라우드 환경의 공격을 방어할 수 없기에 각 테넌트의 프라이버시를 보호하는 클라우드 가상사설망을 제공하는데 한계가 있다. 이에 본 논문은 SGX를 활용한 클라우드 가상사설망의 보안 강화 기법을 제시한다. SGX가 제공하는 하드웨어 지원의 격리된 실행환경과 격리된 메모리 영역을 이용하여 본 논문이 제시하는 방안은 각 테넌트의 프라이버시를 보호하는 키교환과 패킷처리를 제공한다. 또한 테넌트가 원격에서 의도한 보안 정책에 따라 클라우드 가상사설망이 운용되고 있는지 확인할 수 있는 원격 검증 기술을 제시한다. 본 논문이 제시한 방안의 동작을 확인하고 SGX를 클라우드 가상사설망에 적용했을 때 발생할 수 있는 성능 부하를 측정하기 위해 SGX를 지원하는 실제 환경에서 프로토타입을 구현하여 성능 부하를 측정하였다. 또한 본 논문이 제시한 프로토콜을 정형검증 도구를 이용하여 정형검증을 수행하여 그 안전성을 검증하였다.

또한 SGX를 활용한 클라우드 가상사설망 등이 활용하는 enclave의 마이그레이션이 불가능함을 지적하고, 이를 해결하기 위한 새로운 마이그레이션 익스텐션을 제시하였다. 클라우드 환경에서의 마이그레이션은 클라우드 운용 측면에서 중요한 기능으로, 관리자가 가상머신이 운용 중인 물리적 기기를 다른 기기로 교체하는 기능이다. SGX는 하이퍼바이저 등과 같은 소프트웨어가 enclave가 저장된 메모리 영역을 직접 접근할 수 없도록 설계되어, 하이퍼바이저는 기존의 방식으로 enclave를 마이그레이션할 수 없다. SGX 기능 중 페이지 스와핑 기능을 이용하여 enclave가 저장된 메모리 영역을 암호화하여 하이퍼바이저가 접근할 수 있는 메모리 영역으로 추출할 수 있다. 하지만 페이지 스와핑 기능 시 enclave가 저장된 메모리 영역을 암호화하기 위해 사용하는 암호키는 프로세서가 생성한 고유 암호키로 마이그레이션할 다른 기기에서는 접근할 수 없는 암호키이다. 따라서 페이지 스와핑 기능으로 추출된 암호화된 enclave의 메모리 영역을 마이그레이션할 다른 기기가 복호화할 수 없다. 결국 페이지 스와핑 기능을 통해 enclave를 다른 기기로 마이그레이션하는 것이 불가능하다. 이를 해결하기 위해 본 논문은 마이그레이션용 레지스터, 신규 인스트럭션, 그리고 마이그레이션용 enclave로 구성된 새로운 마이그레이션 익스텐션을 제안하고, 이 익스텐션을 활용한 enclave 마이그레이션 프로세스와 구조를 제시한다. SGX 에뮬레이터를 활용하여 제시한 마이그레이션 익스텐션을 구현하여 제안한 마이그레이션 익스텐션의 동작을 확인하고 성능 부하를 예측하였다.

**핵심 낱말** 클라우드 컴퓨팅, 가상사설망, SGX, 마이그레이션, 키교환, 패킷처리

## Abstract


This dissertation presents a security-enhanced cloud Virtual Private Network (VPN) with Software Guard Extensions (SGX) and enclave migration. Cloud VPN is an essential cloud-based network infrastructure that connects on-premise networks with Virtual Private Cloud (VPC) networks securely. However, in the semi-trusted cloud environment cloud VPN suffers from privacy concerns due to the information

disclosure caused by hypervisor vulnerabilities, malicious cloud management operations, etc. The existing literature has limitations in providing each tenant with the privacy-protected cloud VPN because the weak isolation of executing the VPN service in the shared environment does not defense attacks under the semi-trusted cloud environment. We present SGX-VPN, a security-enhanced cloud VPN with SGX. With the hardware-assisted isolated execution environment and the isolated memory region that SGX supports, SGX-VPN provides each tenant with the privacy-protected key exchange and packet processing. SGX-VPN also provides each tenant with an on-demand functionality to verify the integrity of the running security policies in cloud VPN. We implement a prototype on an actual machine to measure the performance penalties of SGX-VPN. We also evaluate SGX-VPN using a formal analysis tool to prove the security of SGX-VPN.

However, there is still a challenging problem with imposing SGX into cloud VPN because existing SGX-enabled Virtual Machine Managers (VMMs) do not provide live migration of SGX-enabled VMs. This management operation is impossible because the VMM cannot directly access the Enclave Page Cache (EPC) pages where the VM's enclaves reside. 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.

**Keywords** Cloud Computing, Virtual Private Network, Software Guard Extensions, Migration, IKE, IPsec

# Contents

Contents . . . . .	i
List of Tables . . . . .	iv
List of Figures . . . . .	v
	
<b>Chapter 1. Introduction</b>	<b>1</b>
<b>Chapter 2. Preliminaries</b>	<b>4</b>
2.1 IPsec . . . . .	4
2.1.1 Overview . . . . .	4
2.1.2 IKE . . . . .	4
2.1.3 Security Policy (SP) . . . . .	5
2.1.4 IPsec Databases . . . . .	5
2.1.5 Packet Processing in IPsec . . . . .	6
2.2 Software Guard Extensions (SGX) . . . . .	7
2.2.1 Isolated Execution Environment . . . . .	8
2.2.2 SGX Data Structures . . . . .	8
2.2.3 EPC Page Swapping . . . . .	8
2.2.4 Local Attestation. . . . .	9
2.2.5 Remote Attestation. . . . .	9
2.3 Data Plane Development Kit (DPDK) . . . . .	10
2.3.1 Environment Abstraction Layer (EAL) . . . . .	11
2.3.2 Poll Mode Driver . . . . .	11
2.3.3 Memory Management . . . . .	11
2.3.4 Packet Forwarding Algorithms . . . . .	11
2.3.5 Cryptodev . . . . .	11
<b>Chapter 3. Related Work</b>	<b>12</b>
3.1 Privacy in Cloud Computing . . . . .	12
3.1.1 Cryptographic Primitives for Data Privacy . . . . .	12
3.1.2 Data Privacy . . . . .	12
3.1.3 Network based Intrusion Detection System . . . . .	13
3.1.4 Cloud VPN . . . . .	14
3.2 Migration in Cloud Computing . . . . .	16
3.2.1 VM Migration . . . . .	16
3.2.2 Enclave Migration . . . . .	17

3.3	SGX in Cloud Computing . . . . .	18
3.3.1	Running Unmodified Applications in SGX . . . . .	18
3.3.2	Distributed Computations with SGX . . . . .	19
3.3.3	SGX in Networking . . . . .	20
<b>Chapter 4.</b>	<b>SGX-VPN: Security-Enhanced Cloud VPN with SGX</b>	<b>22</b>
4.1	Problem Definition . . . . .	22
4.1.1	System Models . . . . .	22
4.1.2	Threat Models . . . . .	22
4.1.3	Goals . . . . .	23
4.2	Design . . . . .	23
4.2.1	Overview . . . . .	23
4.2.2	Terminologies . . . . .	23
4.2.3	SGX-VPN Architecture . . . . .	23
4.2.4	SKE: SGX-enabled Key Exchange for Cloud VPN . . . .	24
4.2.5	SPP: SGX-enabled Packet Processing for Cloud VPN . .	27
4.3	Evaluation . . . . .	30
4.3.1	Analysis . . . . .	30
4.3.2	Comparison . . . . .	32
4.3.3	Efficiency in SA management . . . . .	37
4.3.4	Performance . . . . .	38
<b>Chapter 5.</b>	<b>eMotion: An SGX Extension for Migrating Enclaves</b>	<b>42</b>
5.1	Problem Definition . . . . .	42
5.1.1	System Models . . . . .	42
5.1.2	Threat Models . . . . .	42
5.1.3	Goals . . . . .	42
5.2	Design . . . . .	43
5.2.1	Overview . . . . .	43
5.2.2	SGX Extension for Key Exchange with Remote Attestation . . . . .	43
5.2.3	SGX Extension for Secure Eviction and Loading for Migration . . . . .	46
5.2.4	eMotion Architecture . . . . .	50
5.3	Implementation . . . . .	50
5.3.1	OpenSGX . . . . .	50
5.3.2	eMotion on OpenSGX . . . . .	51
5.3.3	Implementation Result . . . . .	51

5.4	Evaluation . . . . .	53
5.4.1	Analysis . . . . .	53
5.4.2	Comparison . . . . .	53
5.4.3	Performance . . . . .	54
5.5	Possible Deployments . . . . .	56
<b>Chapter 6.</b>	<b>Concluding Remark</b>	<b>57</b>
6.1	Conclusion . . . . .	57
6.2	Future Work . . . . .	57
	<b>Bibliography</b>	<b>59</b>

## List of Tables

4.1	Notations for SGX-VPN . . . . .	24
4.2	Functions of $e_K$ . . . . .	25
4.3	Comparison with the existing approaches; ○ - support, × - not support, N/A - Not Available. . . . .	36
4.4	Additional SAs and key exchange messages for packets within the VPC network; $N$ is a number of VMs. . . . .	38
4.5	Packet processing that $e_P$ supports for SPP V1 . . . . .	39
4.6	Throughput of SPP V1 . . . . .	39
4.7	Packet processing that $e_P$ supports for SPP V2 . . . . .	40
4.8	Comparison of Average Latency( $\mu s$ ) . . . . .	41
5.1	Comparison with the existing migration schemes for SGX enclaves; ○ - support, × - not support. . . . .	54
5.2	The number of instructions in ME and QE during key exchange with remote attestation . . . . .	54
5.3	Elapsed time for secure eviction and loading for migration . . . . .	55



## List of Figures

2.1	Outbound packet processing in IPsec . . . . .	6
2.2	Inbound packet processing in IPsec . . . . .	6
2.3	Overview of SGX; EWB ( $\rightarrow$ ), ELDU/B ( $\rightarrow$ ) . . . . .	7
2.4	Local attestation in SGX based on [1] . . . . .	9
2.5	Remote attestation in SGX based on [1] . . . . .	10
2.6	DPDK architecture . . . . .	10
3.1	Architecture of Protego[2] . . . . .	14
3.2	Possible architecture of sVPN[3] for cloud VPN . . . . .	15
3.3	Configurations of the existing cloud VPNs[4–6] . . . . .	16
3.4	Conceptual diagram of Intel’s patents [7,8] . . . . .	18
4.1	System and Threat Models . . . . .	22
4.2	SGX-VPN architecture; provisioning $C$ , $CFG$ and $SP_i$ ( $\rightarrow$ ), establishing $SA_I$ and $SA_i$ ( $\rightarrow$ ), loading $SA_i$ ( $\rightarrow$ ), and transmitting encrypted packets ( $\rightarrow$ ) . . . . .	24
4.3	Packet processing using $e_P$ ; packet flows ( $\rightarrow$ ) and invocations of $e_P$ ( $\rightarrow$ ) . . . . .	28
4.4	Scyther script for Algorithm 1 . . . . .	31
4.5	Analysis result of Algorithm 1 . . . . .	32
4.6	Scyther script for Algorithm 2 . . . . .	33
4.7	Analysis result of Algorithm 2 . . . . .	34
4.8	Scyther script for Algorithm 3 . . . . .	34
4.9	Analysis result of Algorithm 3 . . . . .	35
4.10	Scyther script for Algorithm 4 . . . . .	35
4.11	Analysis result of Algorithm 4 . . . . .	35
4.12	Scyther script for Algorithm 7 . . . . .	36
4.13	Analysis result of Algorithm 7 . . . . .	37
4.14	SA management in BASE and SGX-VPN . . . . .	37
4.15	A prototype of SPP . . . . .	38
4.16	Comparison of Throughput and Packet Per Second (AES128-CBC HMAC-SHA256) . . . . .	40
4.17	Comparison of Throughput and Packet Per Second (AES128-CTR HMAC-SHA256) . . . . .	41
5.1	eMotion; register read/write ( $\rightarrow$ ) and instruction execution ( $\rightarrow$ ) . . . . .	44
5.2	Diagram of key exchange with remote attestation; register read/write ( $\rightarrow$ ) and instruction execution ( $\rightarrow$ ) . . . . .	46
5.3	Flow chart of ESE . . . . .	48
5.4	Flow chart of ESL . . . . .	49
5.5	Diagram of secure eviction and loading for migration; register read/write ( $\rightarrow$ ), and instruction execution ( $\rightarrow$ ) . . . . .	49
5.6	An architecture of eMotion; key exchange with remote attestation ( $\rightarrow$ ), and secure eviction and loading for migration ( $\rightarrow$ ) . . . . .	50

5.7	Implementation result of eMotion in OpenSGX . . . . .	52
5.8	Elapsed time for secure eviction and loading for migration on enclaves; Directory node and Exit node are Tor enclaves in [9] . . . . .	55
5.9	Possible deployments of eMotion to pre-copy approach [10] . . . . .	55
5.10	Possible deployments of eMotion to post-copy approach [11] . . . . .	56





## Chapter 1. Introduction

Virtual Private Cloud (VPC) is one of cloud deployment models that tenants control the network topologies and security configurations of the VPC networks. Unlike other cloud deployment models (e.g., private, hybrid, and public), VPC not only virtualizes servers and applications, but also the underlying networks as well [12]. Thus, VPC is attractive to the tenants because VPC can provide a seamless transition from a proprietary service infrastructure to a cloud-based infrastructure. The tenants can connect to their resources in the VPC networks as if the resources reside in a single VPC network regardless of the actual regions of the resources. Accordingly, single connection points between the on-premise networks and the VPC networks can reduce the cost and complexity in managing the tenants' resources.

Cloud Virtual Private Network (VPN) is an essential cloud-based network infrastructure that connects the on-premise networks with the VPC networks using Internet Protocol security (IPsec) [13]. Major Cloud Service Providers (CSPs) provide tenants with cloud VPNs [4–6] that protect the traffic on the single connection points between the on-premise networks and the VPC networks while isolating the private IP address space where multiple Virtual Machines (VMs) of the different tenants coexist [14].

Though cloud VPN has been increasingly deployed and indispensable in the VPC networks, security administrators are still reluctant to introduce cloud VPN into their own organizations due to various security concerns raised in the semi-trusted cloud environment, such as information disclosure. First, in a multi-tenant environment, attackers could coincidentally access sensitive information (e.g., session keys) that resides in other guest's cloud VPNs due to hypervisor vulnerabilities like CVE-2015-3340[15]. Second, a malicious insider can intentionally access sensitive information owned by a victim's cloud VPN with cloud management operations. For example, if the cloud administrators attempt to obtain sensitive data used in the guest's cloud VPNs, they can achieve it with operations for cloud management. (e.g., taking a snapshot of cloud VPN containing session keys) Third, the attackers can falsify the leaked information in order to control the guest's cloud VPNs. The attackers manipulate Security Policies (SPs), which defines actions on packets, in the snapshot of cloud VPN (e.g., add a rule to forward decrypted packets to the attackers), and activate the malformed snapshot of cloud VPN. Fourth, the key exchange protocol (e.g., Internet Key Exchange (IKE) [16]) that cloud VPN uses has inherent security concerns: the protocol is vulnerable if ephemeral keys used in the protocol are compromised [17]. When accessing the ephemeral keys via the compromised cloud VPN, adversaries can leak valuable information (e.g., session keys, authentication data, etc.) from the following key exchange messages.

Using the information leaked by the aforementioned security problems, adversaries can conduct other attacks like eavesdropping attack, impersonation attack, etc. These attacks inevitably introduce privacy concerns in cloud VPN. In legacy VPN gateways, the privacy concerns pay less attention because the gateways locate and run inside on-premise networks and thus owners can control the gateways completely. However, in cloud VPN, it is essential to address the privacy concerns because tenants delegate the installation and operation of the virtualized VPN gateways to the CSPs.

However, previous work [2, 3] has limitations in providing each tenant with the privacy-protected cloud VPN in the semi-trusted cloud environment. Protego [2] resolves the inefficiency in resource usages by using a shared VM for the key exchange protocol. If cloud VPN adopts sVPN [3], the tenants utilize a shared VPN service that a hypervisor provides. This weak isolation, which executes the VPN service in

the shared environment [18], does not defense attacks under the semi-trusted cloud environment. Thus, adversaries can extort sensitive information by leveraging the hypervisor vulnerabilities or conducting cloud management operations maliciously.

Software Guard Extensions (SGX) is a good candidate that addresses the privacy concerns in cloud VPN. SGX creates *enclaves* for applications that protect security sensitive code and data from malicious access. The enclaves reside in a part of DRAM invisible to other software, the Enclave Page Cache (EPC). Because CPU fetches the contents of the enclaves from the EPC in an encrypted form, the enclaves can be protected from external access as well as from probing attacks on the DRAM bus by an insider attacker. This strong isolation [19] prevents higher privileged software (e.g., operating systems, hypervisors, etc.) from accessing sensitive information in the enclaves. Therefore, the hypervisors cannot notice the contents of the enclaves even if it has information leak vulnerabilities.

In this dissertation, we propose SGX-VPN that supports the security-enhanced cloud VPN using SGX in the semi-trusted cloud environment. SGX-VPN leverage enclaves to provide each tenant with the strong isolation of VPN services; key exchange and packet processing. Using attestation, each component in SGX-VPN exchanges sensitive information securely without exposing it to untrusted parties. Besides, SGX-VPN provides each tenant with an on-demand functionality to verify the integrity of the running SPs in cloud VPN.

SGX-VPN consists of the algorithms that interchange sensitive information with participating entities securely and verify the integrity of the running SPs remotely. We analyze the algorithms used in SGX-VPN using Scyther [20], one of the formal analysis tools, to prove the security. Scyther is an automated protocol verification tool that supports the verification, the falsification, and the analysis of security protocols. Scyther checks if the security protocols hold the intended security properties (e.g., secrecy). Thus, we leverage this tool for the analysis of SGX-VPN because this tool can prove the secrecy of the transmitted information and the key for integrity protection on SPs.

The followings are the contributions of SGX-VPN.

- **Privacy-Protected cloud VPN.** SGX-VPN supports the privacy-protected cloud VPN by executing VPN services in enclaves that guarantee the strong isolation [19]. SGX-VPN performs the key exchange protocol using the enclave to prevent adversaries from notifying the sensitive information of cloud VPN. SGX-VPN processes packets inside the enclave in order that the adversaries cannot eavesdrop packets transmitted in the VPC network.
- **Architecture based on SGX-VPN.** SGX-VPN presents an architecture to show the practical deployment of SGX-VPN. The architecture protects packets not only between the on-premise network and the VPC network but also within the VPC network.
- **Prototype implementation.** We implement a prototype of SGX-VPN to measure the performance overhead in packet processing. The prototype leverages Data Plane Development Kit (DPDK) [21] to use the enclave for processing packets in the user privilege.

There is still a challenging problem with imposing SGX into cloud VPN: *the existing SGX-enabled VMMs[22] 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 [23] 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 [7, 8] 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.* [24] presented a secure enclave migration in a self-migration manner. Alder *et al.* [25] proposed an enclave migration mechanism that guarantees the consistency of persistent state including sealed data and monotonic counters 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 in 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 Thread Control Structure (TCS). Third, this mechanism cannot migrate the running enclave because the eviction is only applicable to the stopped enclave.

In this dissertation, 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 to the designated enclave and the SGX-enabled processor can access this key.

The followings are the contributions of *eMotion*.

- **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[9], an open source SGX emulator, to demonstrate the operations of *eMotion* and to estimate the impact on enclave migration.



## Chapter 2. Preliminaries

### 2.1 IPsec

IPsec is a secure network protocol suite that supports network-level peer authentication, data-origin authentication, data integrity, data confidentiality (encryption), and replay protection [13]. IPsec mainly consists of security protocols (Encapsulating Security Payload (ESP) [26] and Authentication Header (AH) [27]), key management (Internet Key Exchange (IKE) [16]), and cryptographic algorithms for authentication and encryption. In this section, we explain essential features defined in IPsec for SGX-VPN, and the details can be found in [13, 16, 26, 28].

#### 2.1.1 Overview

IPsec provides two security services in the network layer for processing IP packets: AH [27] and ESP [26]. AH supports integrity and data origin authentication along with optional anti-replay features. Whereas, ESP provides confidentiality to the packets in a company with integrity, data origin authentication, and optional anti-replay features.

IPsec operates in transport mode and tunnel mode that define methods to transform original IP packets. In transport mode, IPsec encrypts and/or authenticates only the payload of the IP packet so that this mode does neither modify nor encrypt the IP header. In tunnel mode, IPsec encrypts and/or authenticates the entire IP packet, and then encapsulates into a new IP packet by adding a new IP header to the ESP or AH header.

The existing cloud VPNs [4–6] support IKE version 2 [16], choose pre-shared key as the authentication method, and encrypt the transmitted packets using ESP tunnel mode.

#### 2.1.2 IKE

IKE [16] is an authenticated key exchange protocol that provides automatic key management for IPsec. IKE establishes and maintains Security Associations (SAs). IKE supports various authentication method such as X.509 certificates, pre-shared key, Extensible Authentication Protocol (EAP), etc. IKE leverages a Diffie-Hellman (DH) key exchange to generate a shared session secret from which cryptographic keys are derived.

An SA is a secure communication session established between two IKE peers and consists of two SAs: IKE SA and IPsec SA. The SA usually defines cryptographic algorithms and contains keys that the cryptographic algorithms use to protect data (IKE messages and inbound/outbound packets).

IKE SA is an SA that stores the master key for creating IPsec SA and protects IKE messages such as IKE\_AUTH, CREATE\_CHILD\_SA, INFORMATIONAL, etc. IKE SA negotiates four algorithms: an encryption algorithm, an integrity protection algorithm, a DH group, and a pseudo-random function (PRF). IKE SA generates SKEYSEED using Equation 2.1.

$$SKEYSEED = \text{PRF}(N_i | N_r, g^{ir}) \quad (2.1)$$

where  $N_i$  and  $N_r$  are the nonces exchanged during the IKE\_SA\_INIT exchange, and  $g^{ir}$  is the shared

secret from the ephemeral DH exchange.

Using SKEYSEED, IKE SA calculates other secrets using Equation 2.2. Other secrets include  $SK_d$  that is the master key for IPsec SA,  $SK_{ai}$  and  $SK_{ar}$  that are keys for the integrity protection on subsequent IKE messages,  $SK_{ei}$  and  $SK_{er}$  that are keys for encrypting all subsequent IKE messages, and  $SK_{pi}$  and  $SK_{pr}$  that are materials to generate the authentication data.

$$\{SK_d|SK_{ai}|SK_{ar}|SK_{ei}|SK_{er}|SK_{pi}|SK_{pr}\} = \text{PRF}(SKEYSEED, N_i|N_r|SPI_i|SPI_r) \quad (2.2)$$

where  $N_i$  and  $N_r$  are the nonces exchanged during the IKE\_SA\_INIT exchange, and  $SPI_i$  and  $SPI_r$  are security parameter indexes (SPIs) for IKE SA.

IPsec SA is an SA that contains the negotiated session protocol, encryption and authentication algorithms, session keys and other session parameters. The session keys for IPsec SA are generated using Equation 2.3.

$$KEYMAT = \text{PRF}(SK_d, N_i|N_r) \quad (2.3)$$

where  $N_i$  and  $N_r$  are the nonces exchanged during the IKE\_SA\_INIT exchange or the fresh nonces exchanged during CREATE\_CHILD\_SA exchange.

To enable VPN connections, it is necessary to use a pair of IPsec SAs (outbound and inbound) for bi-directional communication between two IPsec-enabled peers. IPsec uses a 32-bit SPI along with the destination address in an IP packet header to uniquely identify each IPsec SA for each direction.

The security that IKEv2 guarantees starts from SKEYSEED and the security of SKEYSEED depends on the shared secret ( $g^{ir}$ ) from the ephemeral key. Because  $N_i$  and  $N_r$  are exchanged as plaintext forms, if adversaries can access this shared secret from the ephemeral key, the adversaries can collapse the entire security of IKEv2.

### 2.1.3 Security Policy (SP)

An SP is a rule that specifies behaviors to process specific IP packets. Each SP usually contains source and destination addresses, transport protocol (TCP, UDP), source and destination ports, and actions (BYPASS, DISCARD, PROTECT). If a received IP packet is the target of BYPASS, IPsec forwards the packet to its intended destination. If a received IP packet is the target of DISCARD, IPsec simply drops the packet. If a received IP packet is the target of PROTECT, IPsec processes the packet by using the information defined in the SP: a suitable IPsec SA, etc.

Because SPs define the actual actions performed on all inbound and outbound packets, it is essential to guarantee the integrity of the running SPs. If adversaries can falsify the running SPs, the adversaries can control the actions performed on each packet maliciously. For example, if an adversary inserts the rule to forward the decrypted packets to the adversary's machine, the adversary can sniff sensitive information without extorting session keys.

### 2.1.4 IPsec Databases

IPsec supports three databases : SP database (SPD), SA database (SAD), and peer authorization database (PAD). SPD specifies the policies that apply to all inbound and outbound IP packets. SAD contains parameters that are associated with each SA. PAD provides information such as authentication protocol and data to authorize IKE peers, and thus links between the SPD and IKE.

The SPD is an ordered linked list of entries that contains rules for outbound and inbound packets subject to IPsec protection (PROTECT), be discarded (DISCARD), and be bypassed (BYPASS). For packets subject to IPsec protection, the SPD specifies the IPsec SA that is used to process the packets.

The SAD is a linked list of entries that contains parameters for each SA, and entries in the SPD point to each entry in the SAD. Regarding inbound packets, IPsec uses either SPIs alone or in conjunction with the IPsec protocol type in the packets to look up an SA. Each entry in the SAD has the following data: SPI, cryptographic algorithm and its related parameters (session key, mode, initialization vector (IV), etc.), lifetime of this SA, IPsec protocol mode (AH or ESP), tunnel header IP source and destination address (applicable only to tunnel mode), etc.

The PAD provides functions to links between the SPD and IKE. To this end, the PAD identifies peers authorized to communicate via IPsec, specifies the authentication protocol, manages the authentication data for each peer, restrains the types and values of IDs for the assertion in a peer, and so forth. Thus, the PAD maintains entries that contain the IDs and the authentication data for each peer.

These major databases manage critical data in conjunction with supporting essential functions for IPsec. If adversaries extort the information managed by one of the databases, the security that IPsec guarantees can diminish.

### 2.1.5 Packet Processing in IPsec

Using SPD and SAD, IPsec processes inbound and outbound packets as configured by the security administrator and negotiated with other IKE peer.

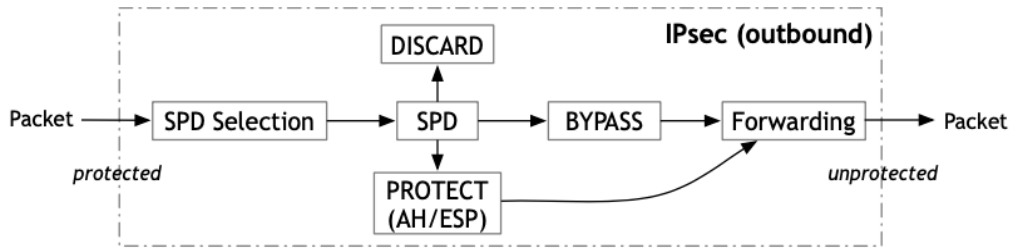


Figure 2.1: Outbound packet processing in IPsec

IPsec processes outbound packets (protected-to-unprotected) as depicted in Figure 2.1. When a packet arrives from the protected interface, IPsec checks if a matching SP exists in SPD using the packet headers. If finding a match, IPsec processes the packet as specified in the matching SP (i.e., BYPASS, DISCARD or PROTECT using AH or ESP). If IPsec should perform AH or ESP on the packet, IPsec uses the SA linked to the matching SP to encrypt and/or authenticate the packet. Finally, IPsec forwards the processed packet to the unprotected interface if the packet is the target of BYPASS or PROTECT.

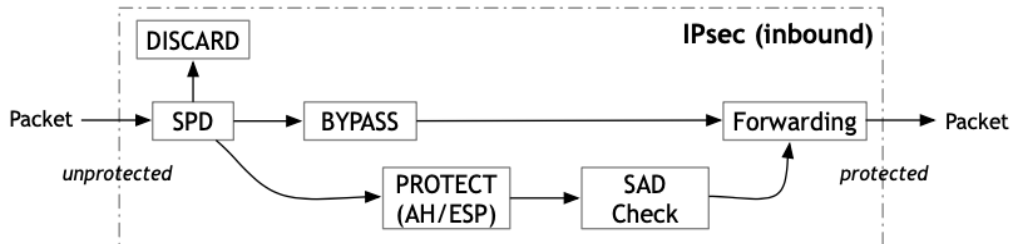


Figure 2.2: Inbound packet processing in IPsec

Inbound processing is quite different from outbound processing because IPsec can utilize SPI to lookup the SA used in incoming packets. IPsec processes these inbound packets (unprotected-to-protected) as depicted in Figure 2.2. When a packet arrives from the unprotected interface, IPsec examines if the packet needs an SAD lookup (AH or ESP) or is the target of BYPASS/DISCARD. If the packet is the target of BYPASS/DISCARD, IPsec processes the packet as defined action (BYPASS/DISCARD). If the packet is the target of the SAD lookup and IPsec finds the matched SA using the SPI in the packet, IPsec authenticates and/or decrypts the packet. Then, IPsec verifies if the processed packet is appropriate for the matched SA by matching the packet against the inbound selectors. If this verification fails, IPsec discards the processed packet.

## 2.2 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 SGX-VPN and eMotion and the SGX details can be found in [1, 23, 29, 30].

Figure 2.3 depicts the overview of SGX with respects to isolated execution environment (2.2.1), SGX data structures (2.2.2), and EPC page swapping (2.2.3).

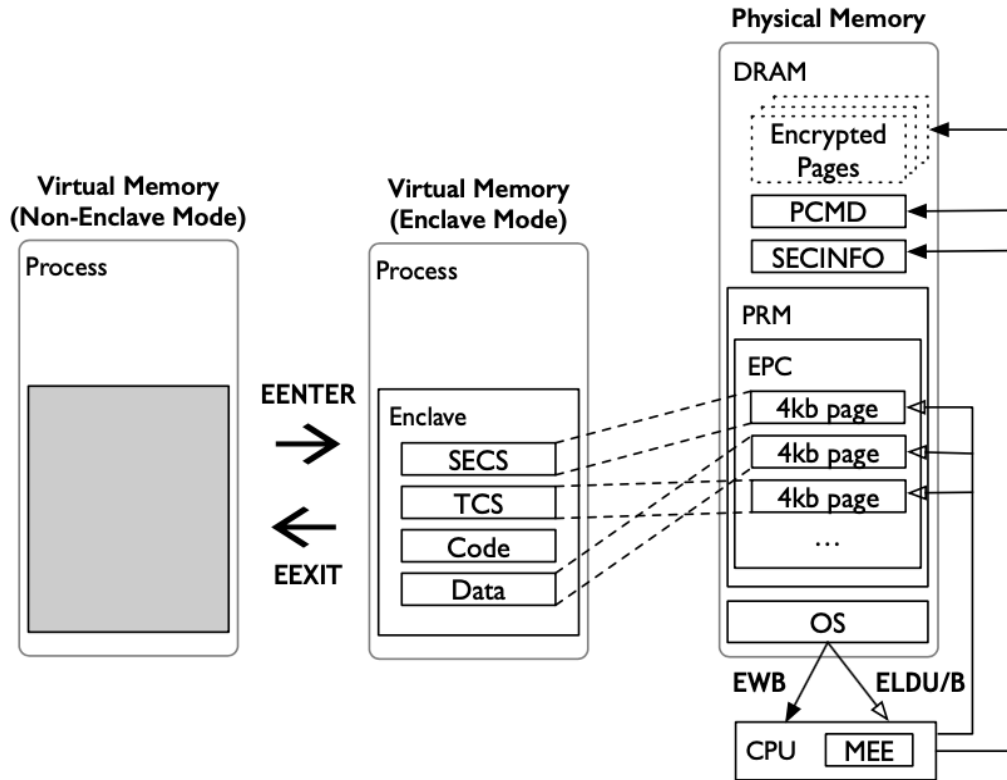


Figure 2.3: Overview of SGX; EWB ( $\rightarrow$ ), ELDU/B ( $\rightrightarrows$ )



### 2.2.1 Isolated Execution Environment

SGX supports a secure container called an enclave that executes code in an isolated environment. When an application process executes an enclave by calling `EENTER`, the context switching happens and the enclave can access the memory area including the memory of the process. Meanwhile, when the enclave finishes its execution by calling `EEXIT`, the context switching also happens and the memory of the enclave cannot be visible to the process and also other software components. Thus, the enclave is isolated from all the other software including its application process on the computer. SGX realize this isolation by leveraging the isolated memory area, the EPC, and adding the Memory Encryption Engine (MEE) to the processor's uncore for protecting the EPC against physical attacks.

The MEE[31] is a hardware unit that protects the confidentiality, integrity, and freshness of the traffic communicated between CPU and EPC, the part of DRAM via the memory bus. To this end, the MEE leverages an integrity tree, the cryptographic primitives for encryption, the message authentication code (MAC), and the anti-replay mechanism. Cryptographic keys that the MEE uses to encrypt and compute authentication tags over EPC are generated uniformly at random at boot time and never leave the CPU.

Using the MEE and the mechanisms implemented in the SGX-enabled processor, SGX can achieve the security properties of the isolated execution environment.

### 2.2.2 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.

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.

### 2.2.3 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.

#### 2.2.4 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.

Figure 2.4 depicts the flow of local attestation in SGX. When two enclaves establish communication paths, enclave A obtains enclave B's **MRENCLAVE<sub>B</sub>**, which is an identifier for each enclave. Then, enclave A sends a signed **REPORT** destined for enclave B. After enclave B succeeds to verify the received **REPORT**, enclave B sends back a signed **REPORT** destined for enclave A using enclave A's **MRENCLAVE<sub>A</sub>**, which is from the received **REPORT**. Finally, enclave A verifies the signed **REPORT** from enclave B. Using this mechanism, two enclaves can convince that the other enclave exists on the same SGX platform.

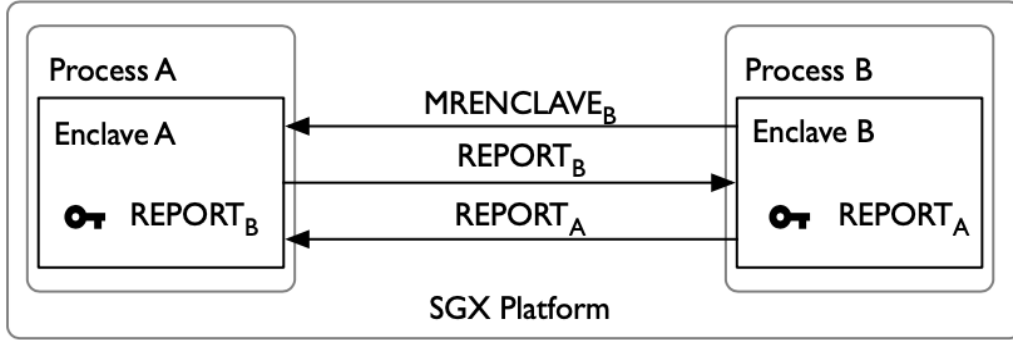


Figure 2.4: Local attestation in SGX based on [1]

#### 2.2.5 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), 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.

Figure 2.5 depicts the flow of remote attestation in SGX. When an enclave and a challenger establish communication paths, the challenger issues a challenge (**Nonce**) to a process running the enclave. Then, the process passes QE's **MRENCLAVE** and **Nonce** to the enclave for local attestation. The enclave sends back a signed **REPORT** destined for QE to the process, and the process forwards **REPORT** to QE for signing. QE returns **QUOTE** for **REPORT** to the process, and the process sends **QUOTE** to the challenger. Using an attestation verification, the challenger verifies **QUOTE** and convinces of the trustworthiness of the enclave and the SGX platform where the enclave is running.

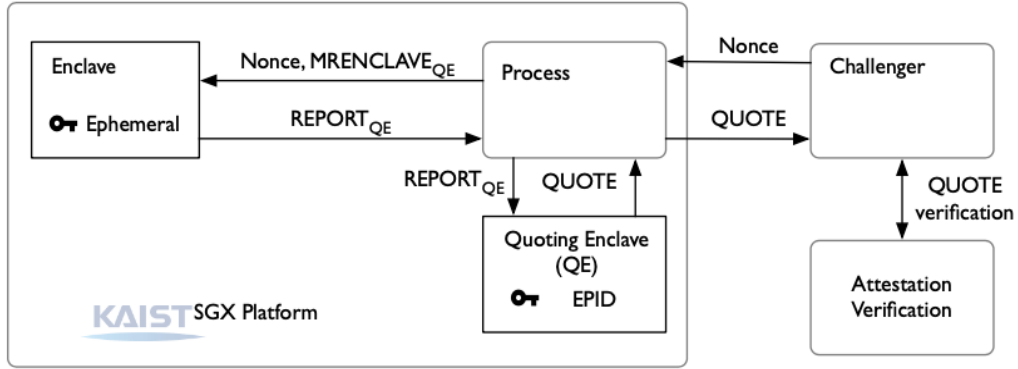


Figure 2.5: Remote attestation in SGX based on [1]

## 2.3 Data Plane Development Kit (DPDK)

DPDK is a framework that supports fast packet processing in dataplane applications running in user-mode [21, 32]. DPDK provides application developers with a set of software libraries and drivers such as environment abstraction layer (EAL), Hash, longest prefix match (LPM), rings, and cryptodev libraries as shown in Figure 2.6.

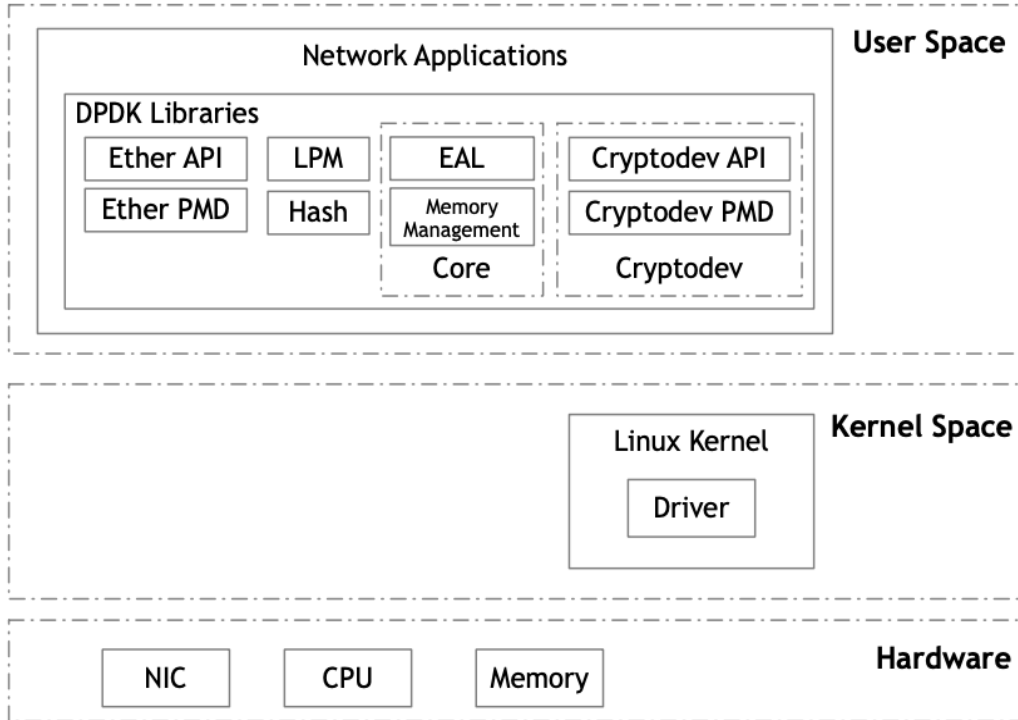


Figure 2.6: DPDK architecture

DPDK supports a run-to-completion model for packet processing, and thus a DPDK application must allocate all necessary resources prior to its main execution, running as execution units on logical processing cores. DPDK does not support a scheduler, and the DPDK application should access all devices via polling in order to reduce the performance overhead imposed by interrupt processing. Besides the run-to-completion model, DPDK provides a pipeline model by passing packets or messages between cores via the rings.

In this section, we explain essential features defined in DPDK for SGX-VPN, and the details can be found in [21, 32].

### 2.3.1 Environment Abstraction Layer (EAL)

The EAL is a core library of DPDK that provides access to low-level resources such as hardware and memory space. The EAL provides the application developers with a generic interface that hides the environment specific resources such as memory space, devices, timers, consoles, etc. Using the EAL, the application developers can utilize the loading and launching of DPDK and their own applications, core affinity/assignment procedures, system memory reservation, etc.

### 2.3.2 Poll Mode Driver

DPDK provides poll mode drivers (PMDs) for all supported network cards and virtual devices. Using the PMDs, DPDK applications can communicate with the network cards directly without the involvement of network drivers.

### 2.3.3 Memory Management

DPDK leverages hugepages to allocate the large memory pool for packet buffers. Using the hugepage allocation, DPDK can increase the performance of packet processing because DPDK needs fewer pages and thus less translation lookaside buffers (TLBs), which reduces the time for translating a virtual page address to a physical page address.

### 2.3.4 Packet Forwarding Algorithms

DPDK includes Hash and LPM libraries to support the corresponding packet forwarding algorithms. Hash library creates a hash table for fast lookup and uses a modified Cuckoo hashing. The hash table is a data structure that contains a set of entries which is identified by a unique key. To increase the performance, the DPDK hash leverages the keys whose sizes are identical throughout the hash table and the size of the keys is determined at the hash creation time. LPM library provides a table search method for 32-bit keys, which is applicable to find the best route match in forwarding IP packets. LPM library uses a variation of the DIR-24-8 algorithm to improve the LPM lookup speed.

### 2.3.5 Cryptodev

A cryptodev is an asynchronous crypto API that supports both hardware and software implementations of crypto poll mode drivers. The cryptodev library in DPDK provides a crypto device framework for managing and provisioning of cryptodev. The cryptodev provides cipher, authentication, chained cipher/authentication, and asymmetric operations.

The cryptodev offers a burst oriented asynchronous API set to schedule crypto operations for DPDK applications. The cryptodev supports enqueue and dequeue burst APIs for the DPDK applications to place and process the crypto operations.

## Chapter 3. Related Work

### 3.1 Privacy in Cloud Computing

Privacy concerns raised by information disclosure has been considered to be of paramount importance in cloud computing. Numerous approaches have been presented to protect the data privacy because tenants entrust the CSPs to store and process their sensitive data. (e.g., health records [33]) However, previous work pays less attention to the privacy concerns in cloud VPN. Instead, the existing literature of cloud VPN [2,14] mainly focuses on the efficiency of resource usages and the scalability for multitenancy. Regarding the security issues in the network of cloud computing, various approaches using Network-based Intrusion Detection System (NIDS) have been proposed [34].

#### 3.1.1 Cryptographic Primitives for Data Privacy

To protect the data privacy in cloud computing, the existing schemes provide the confidentiality of the sensitive data by leveraging Attribute-Based Encryption (ABE) [35–37], Searchable Encryption (SE) [38,39], Predicate Encryption (PE) and Hierarchical Predicate Encryption (HPE) [40,41], Identity-Based Encryption (IBE) [42,43], Proxy Re-Encryption (PRE) [44], and (Fully) Homomorphic Encryption (FHE) [45].

ABE [35] encrypts and decrypts the messages on the basis of user attributes. ABE allows users to selectively share the encrypted data and also supports fine-grained access on the data. Key policy ABE (KP-ABE) [36] and Ciphertext policy ABE (CP-ABE) [37] are the main extensions of the ABE. In KP-ABE, the users can decrypt the encrypted data only if the data attributes satisfy the defined access structure. CP-ABE uses the access policy that defines the access structure to encrypt the data, whereas the users' private keys are generated over a set of attributes.

SE based on symmetric key cryptography and public key cryptography have been presented [38,39]. These cryptographic algorithms perform search operations over the encrypted data without revealing the information about the contents. PE [40] features the fine-grained access control over the encrypted data by associating a ciphertext with descriptive attributes and also associating a secret key with a predicate. The security of the predicate guarantees that an adversary cannot learn anything about the attributes from the ciphertext. HPE [41] delegates the search capabilities only to the users whose secret keys satisfy the predicate. This allows only authorized users to decrypt the encrypted data. IBE [42,43] utilizes any string (e.g., a name, an e-mail address, etc.) as the public key and a trusted party issues the corresponding decryption key. PRE [44] allows a semi-trusted proxy to transform the ciphertext encrypted by one user into a ciphertext that can be decrypted by the other user without any decryption. FHE [45] allows a computation (addition or multiplication) on ciphertexts that produces an encrypted result without any decryption, and the decryption on the encrypted result matches to the result of the computation on the plaintexts.

#### 3.1.2 Data Privacy

Numerous approaches have been proposed to address the privacy concerns in cloud computing with the respect to data privacy [46–53].



Bahrami et al. [46] proposed a lightweight privacy-preserving method to store data on clouds using pseudo-random permutation instead of leveraging cloud computing resources for encryption. Yi et al. [47] presented a solution that allows a user to outsource a mining task to semi-honest servers that perform association rule mining on the encrypted data in the cloud and return encrypted association rules to the user in a vertically distributed environment. Zhou et al. [48] introduced a secure and efficient privacy-preserving dynamic medical text mining and image feature extraction scheme in cloud-assisted e-healthcare systems based on an efficient privacy-preserving fully homomorphic data aggregation, an efficient privacy-preserving function correlation matching from dynamic medical text mining, and a privacy-preserving medical image feature extraction. Pasupuleti et al. [49] proposed an efficient and secure privacy-preserving approach for outsourced data of resource-constrained devices in the cloud computing based on a probabilistic public key encryption algorithm for the encryption and ranked keyword search over the encrypted data for the file retrieval. Rong et al. [50] presented a set of secure building blocks and outsourced collaborative k-Nearest Neighbor (kNN) protocol for the privacy-preserving distributed databases and kNN query along with the concealment of access patterns in the semi-honest model. Consolidated IDentity Management (CIDM)[51] has been proposed to address the security problems of the compromised IDM server, the compromised devices, the network traffic interception. MobiShare [52] provides flexible privacy-preserving location sharing between both trusted social relations and untrusted strangers, and features range query and user-defined access control. Niu et al. [53] presented a caching-based solution to protect location privacy based on an entropy-based privacy metric and new caching-aware dummy selection algorithms.

The aforementioned encryption primitives encrypt the sensitive data before the data is published to cloud computing. This encryption can prevent adversaries from accessing the published data easily. Depending on the used primitives, only authorized users can access the published data or it is feasible to process (search or compute) over the encrypted data.

### 3.1.3 Network based Intrusion Detection System

Several approaches based on NIDS [54–56] have been proposed to secure both external and internal networks of the cloud.

Leu et al. [54] presented Grid Intrusion Detection System (GIDS) that uses grid computing resources to detect strong DDoS attacks. To balance detection loads, GIDS leverages Score Subtraction Approach (SSA) and Score Addition Approach (SAA). For the effective intrusion detection, GIDS adopts two-phase packet detection process; the first phase detects logical and momentary attacks and the second phase detects chronic attacks.

Lo et al. [55] proposed a framework of cooperative IDS to reduce the impact of DDoS attacks in cloud computing. The proposed framework allows IDSs in the cloud computing regions to exchange alert messages, and each of IDSs launches a cooperative agent for computing and determining the acceptance of each of the alert messages sent from other IDSs. Using the proposed framework, the IDSs in the cloud computing regions can prevent the same type of DDoS attack from happening proactively.

Mazzariello et al. [56] presented a misuse detection in open source Eucalyptus cloud environment [57] using Snort [58]. Instead of deploying multiple instances of an IDS within cloud computing, this approach places a single Snort at the frontend cloud controller, which manages cloud instances, for detecting intrusions from the external networks.

### 3.1.4 Cloud VPN

Protego[2] presents a new architecture of distributed IPsec gateway for multitenancy that achieves high availability and elasticity. Protego separates control plane (IKE) and data plane (ESP) and consists of Gateway Management Node (GMN) for handling IKE messages, a set of Gateway Processing Node (GPN) for processing ESP packets, and software load balancers for forwarding the traffic and limiting the bandwidth of each tunnel. This architecture can resolve the inefficiency of resource usages in IPsec gateway VMs while providing elasticity, scalability, high availability, tunnel migration without throughput degradation, and tunnel performance isolation. However, the tenants should entrust the CSPs because a shared VM (GMN) stores all session keys for GPNs as shown in Figure 3.1. Thus, in the semi-trusted cloud environment, if compromising the shared VM, adversaries can not only leak the session keys but also sniff all packets of all tenants.

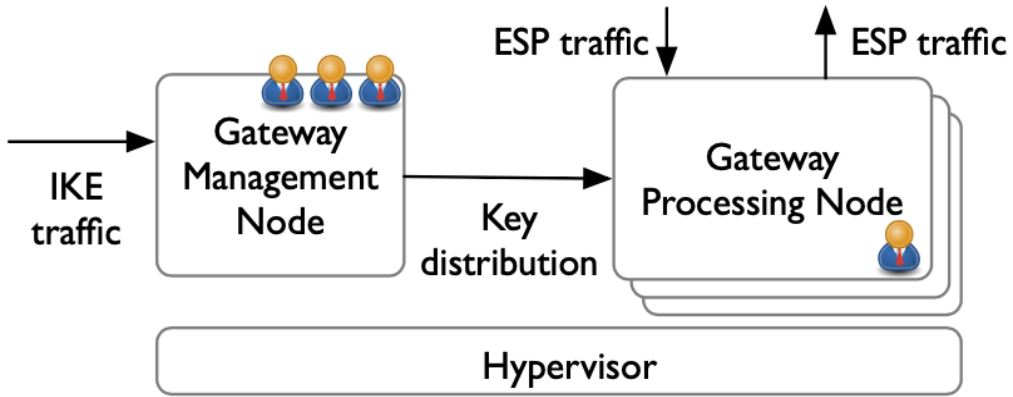


Figure 3.1: Architecture of Protego[2]

Arashloo *et al.* [14] presents a hybrid data plane that enables scalable implementation of network functions including the VPN gateway functionalities at the edge of the cloud. The proposed hybrid architecture leverages a commodity server and a commodity switch to realize high port density with high switching capacity and tunnel lookup tables with around a few million entries. The commodity switch provides the high port density and forwards packets at high speed. The VPN gateway functionalities are realized in the commodity switch and the commodity server. When the switch sends packets from on-premise networks of tenants to the server, the server utilizes DPDK libraries to maintain the internal tunnel lookup table, process packets and return the packets to the switch for forwarding to its destination.

The approaches to enhance the security of VPN has been proposed in [3, 59, 60]. These existing approaches leverage hypervisor or Trusted Platform Module (TPM) [61] as a trust anchor to provide secure IPsec services.

In [3], a VPN architecture for trusted platforms called sVPN has been presented. Based on a microkernel-based operating system, sVPN uses a hypervisor as a trusted and isolated execution environment to execute IKE and IPsec. Using the shared VPN services that the hypervisor provides along with the security configurations, sVPN provides multiple isolated userspace environments with dedicated logical IPsec services with different policy enforcement. If cloud VPN adopts sVPN as shown in Figure 3.2, this weak isolation used in sVPN cannot defend against adversaries in the semi-trusted cloud environment. Thus, if the adversaries compromise the hypervisor that provides tenants with IKE and IPsec, sVPN suffers from the leakage of the session keys and all packets of all tenants.

In [59], an extension of IKEv2 to exchange attestation data has been proposed. Using the proposed

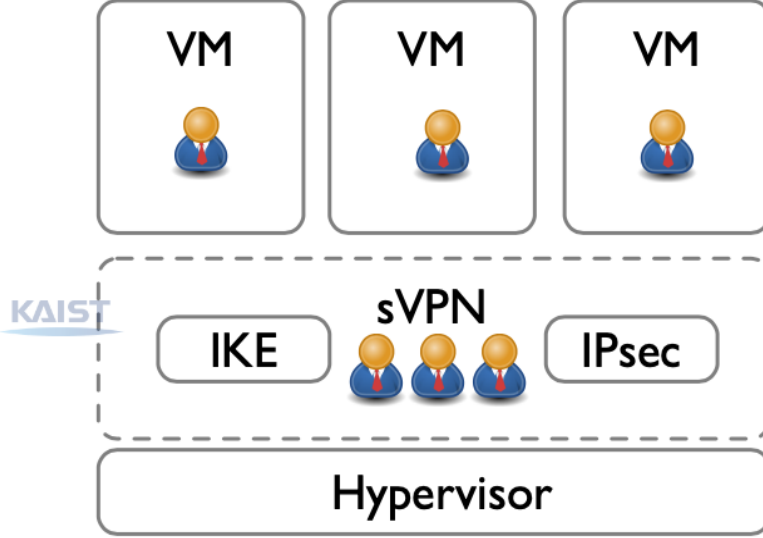
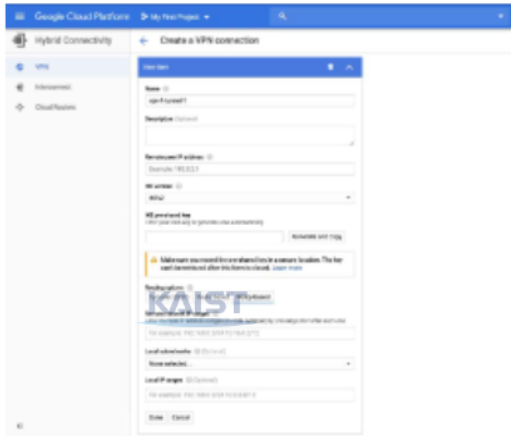


Figure 3.2: Possible architecture of sVPN[3] for cloud VPN

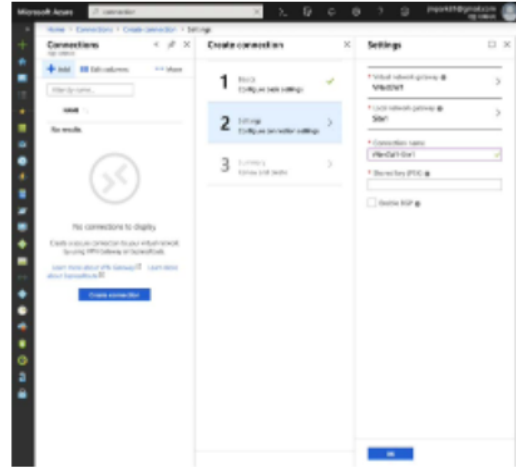
extension, IKE peers can evaluate the internal states of remote peers automatically. The proposed extension introduces a new IKE payload, Attestation Data Payload (ADP), and a new key, Attestation Key (AK). ADP is an IKE payload to transport attestation data for remote attestation, and AK is a shared key to calculate the attestation data for remote attestation. After completing the initial exchange of IKEv2, two IKE peers exchange the ADP payloads in the following IKE exchanges if the peers select the remote attestation algorithm as a part of the IKE SA negotiation. The proposed extension leverages one of the standard IKE exchanges, the INFORMATIONAL exchange, to exchange the ADP payloads. Using `TPMQuote()` that TPM supports and the shared AK, the IKE peers calculate the attestation data of the ADP payloads. Thus, the proposed extension allows the IKE peers to convince that the running status of other IKE peer is trustworthy or not. However, the proposed extension in [59] necessitates additional IKE messages for remote attestation, and the number of the ADP payloads can increase depending on the TCB size.

In [60], an approach to execute remote attestation in VPN environments has been presented. An access requestor (VPN client, AR) and a Policy Decision Point (PDP) perform the Trusted Network Connect (TNC) [62] handshake with remote attestation via the VPN connection established between the access requestor and a Policy Enforcement Point (VPN gateway, PEP). Depending on the result of remote attestation, PDP makes an access decision and sends access instructions to PEP. Then, PEP configures its packet filter to either allow or forbid the AR's access to the TNC-protected network. This approach can enhance the security of the VPN environments by verifying the internal state of the remote endpoints, ARs. However, guaranteeing the integrity of the running configurations in PEP, which this approach does not cover, is also essential in cloud VPN.

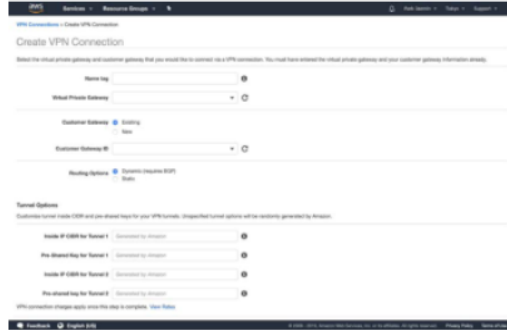
The existing cloud VPNs [4–6] supports the pre-shared key for authentication method of IKE. To configure cloud VPNs, tenants should input the pre-shared keys, which are used in not only their cloud VPNs of the VPC networks but also the legacy VPN gateways located in their on-premise networks, to the CSPs. The tenants also input their intended SPs, which include the policies for their cloud VPNs as well as their legacy VPN gateways, to the CSPs. As shown in Figure 3.3, the tenants should submit the pre-shared keys and the SPs as plaintext forms, and thus the tenants should entrust the CSPs to use cloud VPNs.



(a) Configuration of GCP's cloud VPN



(b) Configuration of Azure's cloud VPN



(c) Configuration of AWS's cloud VPN

Figure 3.3: Configurations of the existing cloud VPNs[4–6]

## 3.2 Migration in Cloud Computing

### 3.2.1 VM Migration

In the cloud environment, VMMs migrate their VMs to different physical machines because of load balancing, fault tolerance, and maintenance. Generally, the source VMM transfers the entire VM's memory pages to the destination VMM via the network until the VMs in the different physical machines are consistent. Accordingly, attackers attempt to capture the transmitted memory pages for extorting the contents of the migrated VM. In the semi-trusted cloud environment, the attackers can compromise software components including VMMs and migration modules to relocate the VM to the compromised environments. Thus, VM migration necessitates high security considerations [63].

Danev et al. [64] identified the security requirements for enabling secure migration of virtual TPM (vTPM) based VMs in private clouds. The authors presented a vTPM key structure suitable for VM-vTPM migration that is compliant with the TPM key usage recommendations, minimizes key regeneration after vTPM migration and prevents vTPM transaction linking. Using the proposed key structure, the author presented a secure VM-vTPM migration protocol.

Wan et al. [65] proposed a secure vTPM migration that utilizes a trusted channel and property-based attestation of the destination platform. With property-based remote attestation, the source and remote hosts mutually verify the integrity and security conditions of the other hosts before migration.



Then the source and destination hosts establish a trusted channel and the source host transfers the VM and vTPM using the established channel securely.

Aslam et al. [66] presented a Trust Token based VM migration protocol that allows a VM migration only if the destination platform is trustworthy. The proposed protocol does not rely on an active (online) trusted third party, rather assumes Platform Trust Assurance Authority (PTAA) as a third party for trust certification. The protocol leverages a Trust Assurance Level (TAL) that specifies the trust level of the cloud platform. The TAL is computed using the TPM credentials, which measures the trust level of hardware, and the Trust Token credential, which specifies the trust level of the software stack. The protocol allows the VM migration and the execution of the migrated VM only if the TAL of the destination platform is in the range of user-specified requirement.

Anala et al. [67] presented a framework for secure live migration of VMs. Using trusted computing, the framework processes attestation and integrity verification of the source and destination hosts. The framework leverages role-based access control policies to avoid VM hopping and useless migrations. The framework also utilizes the encryption and digital signature for the confidentiality and the integrity of the VM transmission. With a per-VM firewall, the framework controls the communication of VM with other components. The framework introduces the host-based firewall and IDS to support network security for the host platform.

### 3.2.2 Enclave Migration

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 [24, 25, 68]. Instead, we refer to Intel's patents [7, 8] as supplementary references.

Our previous work [68] 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.* [24] 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. Because this work is on the basis of the self migration manner, the authors presented only conceptual design suggestions for new SGX instructions. Alder *et al.* [25] proposed an enclave migration mechanism to guarantee the consistency of persistent state including sealed data and monotonic counters. They introduced the possibilities of fork attack and roll-back attack if migrating the persistent state of the enclave has been failed. Then, they presented an improved migration mechanism to use Migration Sealing Key (MSK) for migrating the sealed data and send the monotonic counter values in the migration data.

The mechanisms in [24, 25] necessitates enclaves to use additional libraries that support enclave migration, and thus it is impossible to migrate the enclaves without the specific libraries. This additional effort to use the specific libraries for the enclaves increases the implementation complexity. Moreover, it is impossible for VMMs to suspend the VMs that utilize the enclaves and undergo the VM migration in the self migration manner. Accordingly, it is also a challenging problem to sustain the consistency of the SGX-enabled VMs. Because of these difficulties in cloud management, the self migration manner is not widely used in cloud computing.

Intel presented two patents [7, 8] to enable live migration of SGX-enabled VMs in the managed migration manner. Figure 3.4 depicts the conceptual diagram of Intel's two patents [7, 8]. Intel defines

SGX domain control structure (SDCS) that stores the migration capable keys the controlling enclave generate to be used as a replacement of platform keys. The modified EGETKEY uses the migration capable keys stored in the SDCS in order that a source host can evict enclave pages for migration and a destination host can load the evicted enclave pages for migration. To enable enclave migration, SDCS also contains Version Arrays (VAs) that stores the version numbers (nonces) of the evicted EPC pages and consists of 512 slots of 8 bytes. The source host transmits the SDCS to the destination host via a trusted server or a peer-to-peer connection, and this SDCS protects the enclave pages for secure enclave migration. For this transmission, the controlling enclave in the source host uses a pre-shared key or a secure connection established with the controlling enclave in the destination host. 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. These patents utilize VAs that consume the additional EPC pages and the number of the EPC pages for VAs increases in proportion to the number of the evicted EPC pages

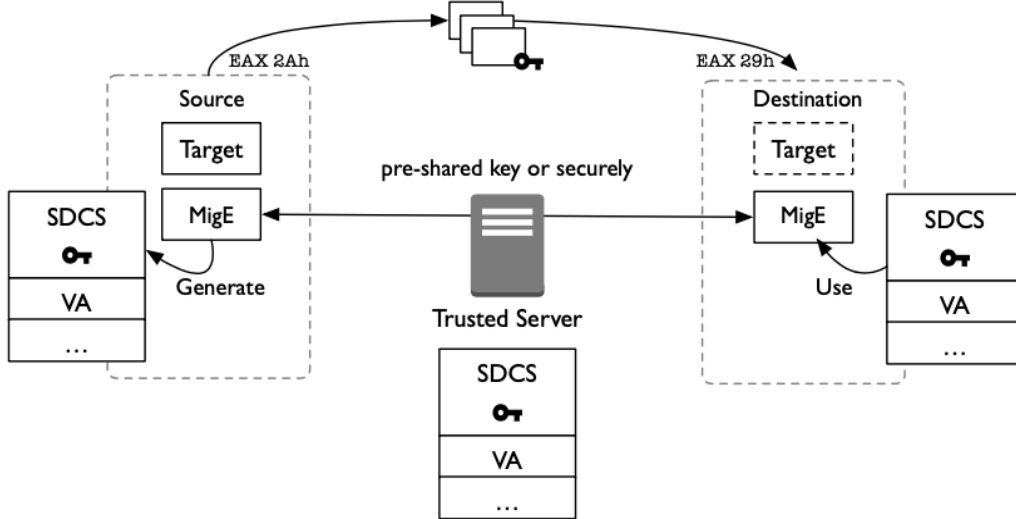


Figure 3.4: Conceptual diagram of Intel's patents [7, 8]

### 3.3 SGX in Cloud Computing

Recently, Microsoft announces its public review of Microsoft Azure Confidential Computing (ACC) that introduces SGX into the cloud platform [69]. IBM also launches the offerings of the SGX-based bare metal servers across all regions on IBM Cloud [70]. With the hardware-assisted isolated execution of applications that SGX guarantee, the CSPs have been undertaking the adoption of SGX into their cloud services for the protection of the tenants' code and data.

#### 3.3.1 Running Unmodified Applications in SGX

Cloud computing provides tenants with shared and configurable computing resources where the tenants' application can execute. This approach can benefit the tenants to run the applications faster with the minimal management costs and the improved manageability. However, security administrators are still reluctant to introduce cloud computing into their own organizations due to various security

concerns, such as information disclosure. To address the security concerns, SGX has been leveraged to run the unmodified tenants' applications securely in cloud computing [19, 71–74].

Haven [71] provides shielded execution of unmodified server applications (e.g., SQL server and Apache) on a commodity OS (Windows). Haven leverages an in-enclave library OS which is derived from Drawbridge [75] to run legacy binary code inside an enclave. Haven also utilizes an SGX remote attestation mechanism to guarantee the security of the running applications without trusting cloud providers.

SCONE [72] is a secure container mechanism for Docker that utilizes SGX to run Linux applications securely in multi-tenant environments. SCONE provides a secure C standard library interface and supports user-level threading and asynchronous system calls for low-performance overhead. The prototypes based on SCONE includes Apache, Redis, NGINX, and Memcached to show the operations of SCONE and measure the performance overhead caused by SCONE.

Graphene-SGX [73] ports Graphene [76] to SGX for a fully-featured library OS where unmodified Linux applications can execute on SGX. Graphene-SGX realizes the security improvements that contain integrity support of dynamically-loaded libraries, enclave-level forking, and secure inter-process communication (IPC). The evaluation of Graphene-SGX includes famous Linux web applications (Apache, Redis, NGINX, and Memcached) and commonly used command-line applications (R, GCC, CURL).

PANOPLY [19] presents a micro-container which is a unit of code and data isolated in SGX enclaves. This micro-container provides commodity Linux applications with the standard POSIX abstractions including access to filesystems, network, multi-threading, multi-processing and thread synchronization primitives. PANOPLY also supports an integrity property for the inter-enclave interactions by enforcing that the execution of the application follows the legitimate control and data-flow. PANOPLY demonstrates 4 real-world Linux applications based on PANOPLY including Tor, H2O, OpenSSL, and FreeTDS.

SGXKernel [74] presents an in-enclave library OS that provides asynchronous cross-enclave communication and preemptible in-enclave multi-threading. These properties of SGXKernel can achieve a switchless design that obviates the needs of enclave transitions.

These SGX-based mechanisms leverage the hardware-assisted isolation and protection of applications' code and data, so the inherent security concerns raised in cloud computing have been addressed. However, without the consideration of enclave migration, these approaches can suffer from management problems such as fault management, load balancing, system maintenance, etc.

### 3.3.2 Distributed Computations with SGX

Running entire legacy applications in enclaves have limitations because of the limited EPC size (128MB) and the large Trusted Code Base (TCB) size [77]. Regarding of distributed computations in cloud computing, it is more efficient to partition the applications in trusted and untrusted parts where the actual data processing tasks are running in the enclaves [77–79].

Brenner et al. [78] presented the ZooKeeper Privacy Proxy (ZPP) that supports a lightweight and transparent encryption layer for ZooKeeper [80]. ZPP runs inside a TEE (e.g., enclaves) to protect data and naming information while mediating client connections.

VC3 [79] supports the secure execution of MapReduce [81] computations in cloud computing while protecting the code and data and ensuring the correctness and completeness of the results. VC3 executes the actual data processing tasks inside enclaves while the underlying framework, Hadoop [82], is unchanged. VC3 also deploys new protocols for secure distributed MapReduce computations. VC3



provides a compiler that enforces region self-integrity invariants for all MapReduce code running within the enclaves to prevent attacks of unsafe memory reads and writes.

SecureKeeper [77] is an enhanced version of the ZooKeeper that uses SGX to preserve the confidentiality and basic integrity of ZooKeeper-managed data. SecureKeeper leverages two enclaves; an entry enclave that protects the client-replica connection and the data security of the ZooKeeper data store, and a counter enclave that processes special write requests of sequential nodes at the leader replica of the ZooKeeper.

### 3.3.3 SGX in Networking

SGX, one of the commoditized Trusted Execution Environments (TEEs), has been considered to address the security and privacy concerns in network applications because many networking applications run on commodity hardware [83]. In [83], SGX shows the possibilities to solve policy privacy issues in software-defined inter-domain routing, secure the Tor anonymity network, and introduce in-network functions (middleboxes) into TLS sessions securely.

SGX has been also utilized to secure the existing network functions in Network Function Virtualization (NFV) [84–89], edge computing [90], and Tor [91]. However, to the best of our knowledge, there is no related work that uses SGX to address the privacy concerns in cloud VPN.

S-NFV [84] identifies security problems raised in the management of the internal states of NFV applications. S-NFV utilizes SGX to securely isolate the state of NFV applications so that malicious hosts and buggy applications cannot access them. As a proof-of-concept, S-NFV uses Snort [58] to demonstrate the operations of S-NFV.

Trusted Click [85] presents an integration of SGX into Click [92] for supporting arbitrary NFV applications securely. Each Click element receives encrypted packets from a gateway that outsources the packet processing to NFV. Then, each Click element decrypts the encrypted packets using a decryption key that the gateway sends it via a secure channel established after remote attestation of the Click element. Finally, each Click element processes the decrypted packets and re-encrypts them before passing to the next element. Trusted Click attempts to demonstrate the usability and the performance of SGX in NFV. However, Trusted Click does not provide the key management of the decryption key and the practical implementation of the NFV applications.

SGX-Box [86] presents a secure middlebox system that supports secure inspection on encrypted traffic by using SGX. SGX-Box protects decrypted payloads and session keys within the SGX enclave, and thus all payloads travel to other middlebox components as encrypted forms. To help middlebox developers, SGX-Box supports SB lang that has characteristics of easy-to-use abstraction and a high-level programming language. As a proof-of-concept, SGX-Box utilizes Intrusion Detection System (IDS). SGX-Box proposes a secure out-of-band key sharing mechanism that allows an end server to share its session keys with an SGX-Box middlebox through an out-of-band channel established via remote attestation.

LightBox [87] provides a system that supports full-stack protected stateful middleboxes at native speed. Using LightBox, enterprises can forward the packets to the SGX-enabled middleboxes securely with all metadata, including low-level packet headers, packet size, count and timestamps. To this end, LightBox introduces a virtual network device (etap) that allows access to fully protected network packets at line rate without leaving the enclave. LightBox uses PRADS [93] that detects network assets in packets based on predefined fingerprints and signatures, and implements a simple IDS that identifies malicious patterns based on a TCP reassembly library libntoh [94].

SafeBricks [88] protects generic network functions (NFs) from an untrusted cloud by executing the NFs within enclaves. SafeBricks builds upon NetBricks [95] that provides a framework to build and execute arbitrary NFs. SafeBricks leverages an architecture that partitions an NF application into enclave code and non-enclave code to reduce TCB size. Also, SafeBricks develops an architecture that uses shared memory where enclave thread and non-enclave thread compute without the need for enclave transitions. Instead of using a separate enclave for each NF, SafeBricks supports chains of NFs within the same enclave and utilizes Rust language to isolate the NFs. SafeBricks uses four different NF applications; firewall, deep packet inspection, network address translation, and load balancer, for the evaluation.

EndBox [89] provides a decentralized system that executes middlebox functions on client machines at the network edge. EndBox combines VPN and middlebox functions that SGX protects to ensure that the middlebox processes all traffic, including encrypted packets. EndBox implements all middlebox functions using the Click modular router [92] in the client machines. EndBox leverages VPN as the only access point to the network, and thus VPN forwards all packets to the middlebox functions for processing. After the middlebox functions complete their processing on the packets, VPN handles the process packets to be transferred to the network in the encrypted forms. EndBox uses four different middlebox functions; load balancing, IP firewall, intrusion detection, and prevention system, and DDoS prevention, for the evaluation.

AirBox [90] supports fast, scalable and secure edge functions (EFs) for device-cloud interactions. AirBox provides back-end driven onloading to the edge that guarantees fast and scalable EF provisioning, integrity of EF execution, and confidentiality of the state stored at the edge.

SGX-Tor [91] leverages SGX to prevent code modifications and to restrict the information disclosure to untrusted parties. SGX-Tor can reduce the power of Tor adversaries to that of a network-level adversary, and thus the adversaries cannot see the internal state of Tor components. To this end, SGX-Tor protects private Tor operations like TLS decryption and circuit demultiplexing from the adversaries.

## Chapter 4. SGX-VPN: Security-Enhanced Cloud VPN with SGX

### 4.1 Problem Definition

#### 4.1.1 System Models

We consider that each tenant  $T$  operates a VPN gateway  $V_T$  in its on-premise network, and  $V_T$  connects to  $V_C$  for managing VM instances in the VPC network as depicted in Figure 4.1. Hypervisors where  $V_C$  and the VM instances run support SGX [22] and DPDK. Thus, the hypervisors run in commodity servers that support the SGX-enabled processors and the DPDK-compatible network interfaces.  $V_T$  operates in the on-premise network with  $T$ 's control, and we assume that  $V_T$  runs as intended. We assume that  $V_C$  and the VM instances connect to the Intel-operated service called Intel Attestation Service (IAS) [96] so that  $T$  launches remote attestation via the protocol that the SGX implementation supports.

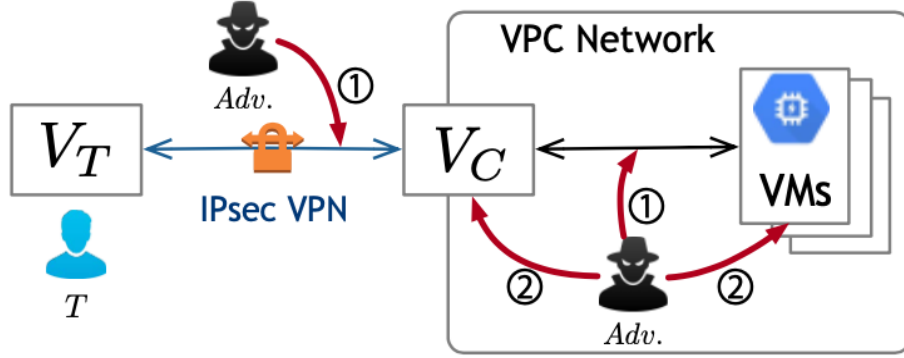


Figure 4.1: System and Threat Models

#### 4.1.2 Threat Models

We assume the semi-trusted cloud environment, and trust only enclaves and the mechanisms implemented in the SGX-enabled processors. In the semi-trusted cloud environment,  $V_C$  and VM instances are usually running as intended, but they can become *malicious* when an attacker  $Adv.$  corrupts them via security attacks or software bugs of hypervisors or cloud VPN. Because  $Adv.$  has full control over the memory and the network resources of the hypervisors or the cloud VPN,  $Adv.$  can read all memory pages of DRAM excluding the EPC and sniff all packets. As depicted in Figure 4.1,  $Adv.$  can sniff packets not only between  $V_T$  and  $V_C$  but also within the VPC network (①).  $Adv.$  also attempts to acquire sensitive information by reading the memory pages that the hypervisors or the cloud VPN access (②).  $Adv.$  can simply discard key exchange messages and/or inbound/outbound packets; such a denial-of-service (DoS) attack is out of scope in SGX-VPN.

### 4.1.3 Goals

To resolve the privacy concerns in cloud VPN, SGX-VPN should support the strong isolation of key exchange and packet processing for each tenant. *Adv.* should not access any cryptographic keys including credentials ( $C$ ), ephemeral keys ( $EK$ ), and session keys ( $SK$ ). *Adv.* should not falsify the SPs that SGX-VPN utilizes to process packets.  $T$  should convince that its intended SPs are running in SGX-VPN indeed. We define the goals for SGX-VPN as follows.

**G1:** Isolated execution of cloud VPN.

SGX-VPN should feature the isolated execution of key exchange and packet processing for each  $T$ . That is, SGX-VPN should guarantee the strong isolation rather than the weak isolation.

**G2:** Secrecy of cryptographic keys.

SGX-VPN should guarantee the secrecy of cryptographic keys including  $C$ ,  $EK$ , and  $SK$  by restricting the access on these keys only to trustworthy parties. Thus, *Adv.* should not access any cryptographic key so as to camouflage as a legitimate entity and sniff key exchange messages and inbound/outbound packets. It is crucial to restrict the access on these keys only to trustworthy parties because *Adv.* can attempt to steal the keys by reading the memory.

**G3:** Integrity protection on SPs.

SGX-VPN should protect the integrity of running SPs and convince  $T$  that SGX-VPN operates its intended SPs without any modification. Thus, *Adv.* should not falsify any SP in SGX-VPN in order that *Adv.* cannot control any packet maliciously. If *Adv.* adds a rule to forward decrypted packets to *Adv.*, *Adv.* can obtain valuable information easily.

**G4:** Protection on packets within the VPC network.

*Adv.* should not extort any content in packets transmitted inside the VPC network. It is essential to encrypt all packets in the VPC network.

## 4.2 Design

### 4.2.1 Overview

SGX-VPN supports privacy-protected key exchange and packet processing by leveraging SGX. SGX-VPN consists of SKE (SGX-based Key Exchange) and SPP (SGX-based Packet Processing), that guarantee the strong isolation of the VPN services for each tenant using enclaves. SKE includes an enclave ( $e_K$ ), and SPP contains an SGX-based DPDK cryptodev ( $C_P$ ) and an enclave ( $e_P$ ).

### 4.2.2 Terminologies

We define the notations for SGX-VPN as depicted in Table 4.1 where  $i$  is an index ( $i = 0, 1, 2, \dots$ ).

### 4.2.3 SGX-VPN Architecture

The SGX-VPN architecture consists of  $V_T$ ,  $V_C$ , and  $T$ 's VMs that enables SPP as depicted in Figure 4.2.  $V_T$  performs remote attestation to  $V_C$  for establishing a secure channel with  $e_K$  of  $V_C$ . Using



Table 4.1: Notations for SGX-VPN

Notation	Description
$SA_I$	Security association for key exchange
$SA_i$	Security association for packet processing
$SP_i$	A set of rules for packet processing
$E(I)_K$	An encryption of input $I$ using $K$
$H(I)$	A hash calculation of input $I$
$K_{LA}$	A session key resulted from local attestation
$K_{RA}$	A session key resulted from remote attestation

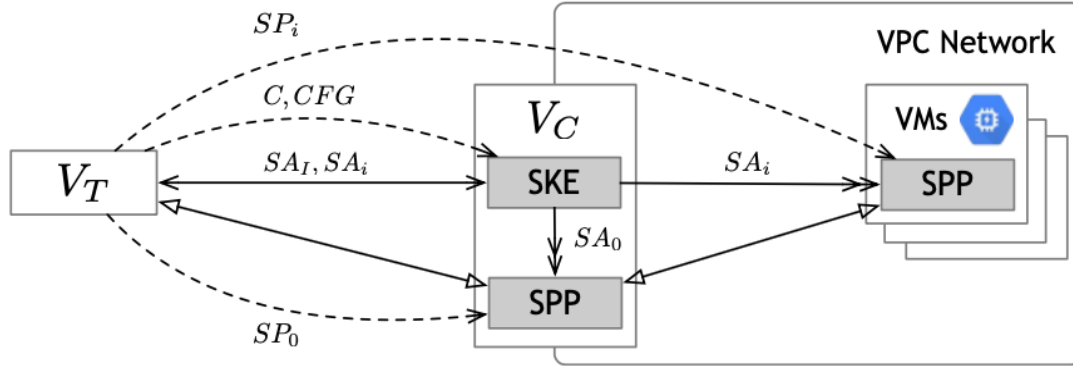


Figure 4.2: SGX-VPN architecture; provisioning  $C$ ,  $CFG$  and  $SP_i$  ( $\dashrightarrow$ ), establishing  $SA_I$  and  $SA_i$  ( $\rightarrow$ ), loading  $SA_i$  ( $\rightarrow$ ), and transmitting encrypted packets ( $\rightarrow$ )

the secure channel,  $V_T$  provisions  $C$  and  $CFG$  to  $e_K$  of  $V_C$ .  $CFG$  defines security configurations that SGX-VPN utilizes to establish  $SA_I$  and  $SA_i$ .

$V_C$  equips with SKE and SPP, and negotiates  $SA_I$  and  $SA_i$  with  $V_T$  using SKE. Each VM instance enables SPP to protect the packets within the VPC network. Instead of implementing SKE into each VM, SGX-VPN operates only one SKE in  $V_C$  to execute key exchange on behalf of each VM. To identify the target VMs,  $V_T$  includes a new IKE payload,  $VMID$ , while executing key exchange. When establishing  $SA_i$ , SKE in  $V_C$  loads  $SA_i$  to SPP in  $V_C$  and in each VM using  $K_{LA}$  and  $K_{RA}$ , respectively.

SPP also needs SPs for processing packets.  $V_T$  provisions  $T$ 's intended SPs ( $SP_i$ ) to SPP using  $K_{RA}$ .  $SP_i$  consist of rules and their order numbers that specify the method for SPP to process the packets. Using  $SA_i$  and  $SP_i$ , SPPs in  $V_C$  and each VM process the packets.  $V_T$  can request the remote verification of the running  $SP_i$  to SPP in  $V_C$  and each VM. For this verification,  $V_T$  verifies the evidence that SPP generates against  $SP_i$  via the remote attestation mechanism.

#### 4.2.4 SKE: SGX-enabled Key Exchange for Cloud VPN

SKE provides the SGX-based key exchange by leveraging  $e_K$  along with remote attestation.  $V_T$  provisions  $C$  and  $CFG$  to  $e_K$  via a secure channel established by remote attestation. Then, SKE establishes  $SA_I$  and  $SA_i$  with  $V_T$  by using  $e_K$  that stores  $C$  and  $CFG$  securely and processes the tasks for the key exchange payloads inside the enclave. This isolated execution of key exchange prevents  $Adv.$  from accessing sensitive information.



### Provisioning $C$ and $CFG$ to $e_K$ .

$V_T$  provisions  $C$  and  $CFG$  to  $e_K$  that runs in  $V_C$ .  $V_T$  performs remote attestation with  $e_K$  to check if  $V_C$  uses the legitimate SGX-enabled processor and  $e_K$  is not altered. After remote attestation succeeds, a secure channel is established between  $V_T$  and  $e_K$ , and  $K_{RA}$  is produced that  $V_T$  uses for encryption.  $V_T$  yields  $E_0$  by encrypting  $C$  and  $CFG$  using  $K_{RA}$ , and calculate  $H_0$  against  $E_0$ . Then,  $V_T$  sends  $E_0$  and  $H_0$  to  $e_K$ , and  $e_K$  verifies  $H_0$  and decrypts them using  $K_{RA}$ . Finally,  $e_K$  loads  $C$  and  $CFG$  to its EPC to prevent other software components from accessing them. Algorithm 1 depicts the protocol for provisioning  $C$  and  $CFG$  to  $e_K$ .

---

#### Algorithm 1 Protocol for provisioning $C$ and $CFG$

---

- 1:  $V_T \leftrightarrow e_K$ : Perform remote attestation  $\rightarrow K_{RA}$
  - 2:  $V_T$ :  $E_0 \rightarrow E(C, CFG)_{K_{RA}}$
  - 3:  $V_T$ :  $H_0 \rightarrow H(E_0)$
  - 4:  $V_T \rightarrow e_K$ :  $E_0, H_0$
  - 5:  $e_K$ : Check  $H_0$  and decrypt  $E_0 \rightarrow C, CFG$
  - 6:  $e_K$ : Load  $C$  and  $CFG$
- 

### Key exchange enclave

$e_K$  is an enclave that provides SKE with the key exchange functions.  $e_K$  equips with the essential features such as secure storage for  $C$  and  $CFG$  and secure operations for the key exchange payloads.  $e_K$  also provides cryptographic functions such as modular exponentiation, random number generation, pseudo-random function (PRF), and symmetric algorithms. Using the isolated execution environment that SGX guarantees,  $e_K$  protects the security sensitive information from  $Adv.$ 's access.  $e_K$  supports three types of functions as denoted in Table 4.2: storage of Provisioned Data (PD), and generation of Dynamic Data (DD) and contents for the Key Exchange Payloads (KEP).

Table 4.2: Functions of  $e_K$ .

	Type	Description
$C$	PD	Secure storage
$CFG$	PD	Secure storage
$EK$	DD	Ephemeral keys
$SK$	DD	Session keys for $SA_I$ and $SA_i$
$SA$	KEP	SA proposals from $CFG$
$KE$	KEP	DH public numbers
$N$	KEP	Nonces
$ID$	KEP	Identifications from $CFG$
$TS$	KEP	Traffic selectors from $CFG$
$AUTH$	KEP	Authentication data using $EK$ and $C$
$ENC$	KEP	Contents in encrypted payload

### Processing key exchange for $V_C$

After Algorithm 1 succeeds,  $V_T$  can connect to  $V_C$  securely.  $V_T$  and SKE in  $V_C$  perform Algorithm 2 to establish  $SA_I$  and  $SA_0$ . This algorithm extends the initial exchange of the existing key exchange protocol [16]. Then, SKE in  $V_C$  loads  $SA_0$  to SPP in  $V_C$  using a secure channel established by local attestation. SKE can support any authentication method, but we use the pre-shared key (PSK) method for simplicity.

---

**Algorithm 2** Key exchange protocol for  $V_C$ 

---

- 1:  $V_T \rightarrow V_C$  :  $SA_{i1}, KE_i, Ni$
  - 2:  $V_C \rightarrow V_T$  :  $SA_{r1}, KE_r, Nr$
  - 3:  $V_T, V_C$  : Establish  $SA_I$
  - 4:  $V_T \rightarrow V_C$  :  $ENC_i \leftarrow E(ID_i, AUTH_i, SA_{i2}, TS_i, TS_r)_{SA_I}$
  - 5:  $V_C \rightarrow V_T$  :  $ENC_r \leftarrow E(ID_r, AUTH_r, SA_{r2}, TS_i, TS_r)_{SA_I}$
  - 6:  $V_T, V_C$  : Establish  $SA_0$
  - 7:  $V_C(SKE) \leftrightarrow V_C(SPP)$ : Perform local attestation  $\rightarrow K_{LA}$
  - 8:  $V_C(SKE): E_1 \rightarrow E(SA_0)_{K_{LA}}$
  - 9:  $V_C(SKE): H_1 \rightarrow H(E_1)$
  - 10:  $V_C(SKE) \rightarrow V_C(SPP): E_1, H_1$
  - 11:  $V_C(SPP)$ : Check  $H_1$  and decrypt  $E_1 \rightarrow SA_0$
  - 12:  $V_C(SPP)$ : Load  $SA_0$
- 

Suppose that  $V_T$  and  $V_C$  act as the initiator and the responder, respectively.  $V_T$  and  $SA_I$  in  $V_C$  exchange the key exchange payloads including SA ( $SA_{i1}, SA_{r1}$ ), KE ( $KE_i, KE_r$ ), and N ( $Ni, Nr$ ) to establish  $SA_I$ . Then,  $V_T$  and  $SA_I$  in  $V_C$  exchange the payloads containing ID ( $ID_i, ID_r$ ), AUTH ( $AUTH_i, AUTH_r$ ), SA ( $SA_{i2}, SA_{r2}$ ), TS ( $TS_i, TS_r$ ) to establish  $SA_0$ . When receiving the payloads, SKE delegates the payload processing to  $e_K$  for generating  $SA_I$  and  $SA_0$ .  $e_K$  generates  $EK, SK, SA_I$ , and  $SA_0$  in the isolated memory where  $e_K$  runs and this sensitive data does not leave  $e_K$ .

Once  $V_T$  establishes  $SA_0$  with SKE in  $V_C$ , SKE executes local attestation to  $e_P$  of  $V_C$ . In this attestation, SKE in  $V_C$  interworks with  $e_K$  to establish  $K_{LA}$  with  $e_P$  of  $V_C$ .  $e_K$  of  $V_C$  produces  $E_1$  by encrypting  $SA_0$  using  $K_{LA}$ , and calculates  $H_1$  against  $E_1$ . Then, SKE in  $V_C$  sends  $E_1$  and  $H_1$  to  $e_P$  of  $V_C$ .  $e_P$  of  $V_C$  loads  $SA_0$  after checking  $H_1$  and decrypting  $E_1$  using  $K_{LA}$ .

**Processing key exchange for SPP of VMs**

SGX-VPN adds SPP to each VM to protect packets within the VPC network. After establishing  $SA_I$  and  $SA_0$  with  $V_C$ ,  $V_T$  establishes additional  $SA_i$  ( $i \geq 1$ ) with SKE in  $V_C$  for SPPs of VMs using the existing  $SA_I$ . Then, SKE in  $V_C$  loads  $SA_i$  to SPP of the VMs using secure channels established by remote attestation.

---

**Algorithm 3** Key exchange protocol for SPP

---

- 1:  $V_T \rightarrow V_C$  :  $ENC_i \leftarrow E(SA_i, Ni, TS_i, TS_r, VMID, [VMID])_{SA_I}$
  - 2:  $V_C \rightarrow V_T$  :  $ENC_r \leftarrow E(SA_r, Nr, TS_i, TS_r)_{SA_I}$
  - 3:  $V_T, V_C$  : Establish  $SA_i$
  - 4:  $V_C \leftrightarrow SPP$ : Perform remote attestation  $\rightarrow K_{RA}$
  - 5:  $V_C: E_i \rightarrow E(SA_i)_{K_{RA}}$
  - 6:  $V_C: H_i \rightarrow H(E_i)$
  - 7:  $V_C \rightarrow SPP: E_i, H_i$
  - 8: SPP: Check  $H_i$  and decrypt  $E_i \rightarrow SA_i$
  - 9: SPP: Load  $SA_i$
- 

Algorithm 3 depicts the protocol that establishes  $SA_i$  for SPP of the VMs. This algorithm extends the CREATE.CHILD\_SA exchange of the existing key exchange protocol [16]. SKE in  $V_C$  includes a new IKE payload (VMID) that indicates the target VMs equipped with SPP. If the key exchange message includes only one VMID, the established  $SA_i$  protect packets between  $V_C$  and SPP of the VM indicated by VMID. Otherwise, the established  $SA_i$  protects packets between two SPPs of the VMs indicated by VMIDs.

When establishing  $SA_i$  for SPPs of the VMs with  $V_T$ , SKE in  $V_C$  executes remote attestation to  $e_P$ s of SPPs. This remote attestation produces  $K_{RA}$  between  $e_K$  of  $V_C$  and  $e_P$  of SPP.  $e_K$  of  $V_C$  yields  $E_i$  by encrypting  $SA_i$  using  $K_{RA}$ , calculates  $H_i$  against  $E_i$ . Then, SKE in  $V_C$  sends  $E_i$  and  $H_i$  to  $e_P$  of SPP.  $e_P$  of SPP loads  $SA_i$  after checking  $H_i$  and decrypting  $E_i$  using  $K_{RA}$ .

#### 4.2.5 SPP: SGX-enabled Packet Processing for Cloud VPN

SPP processes actual packets arriving at  $V_C$  or each VM using the rules defined in the SPs and  $SA_i$  sent from SKE in  $V_C$ .  $V_T$  provisions its intended  $SP_i$  to  $e_P$  of SPP.  $SP_i$  includes the rules to process all PROTECT packets and to allow remote attestation (BYPASS). Thus, SPP simply drops the packets which are not the target of PROTECT excluding remote attestation (BYPASS).

SPP leverages DPDK to process the packets using  $e_P$ . Because enclaves run only in the user space, SPP should leverage this framework for supporting fast packet processing in the user space using  $e_P$ . SPP introduces  $C_P$  that is an SGX-based DPDK cryptodev to invoke  $e_P$ .

##### Provisioning SPs to $e_P$

$V_T$  provisions  $SP_i$  to  $e_P$  of SPP that runs in  $V_C$  or each VM.  $V_T$  performs remote attestation with  $e_P$  to check if  $V_C$  or SPP uses the legitimate SGX-enabled processors and  $e_P$  launched in  $V_C$  or SPP is not forged.  $V_T$  provisions  $SP_i$  to  $e_P$  securely by using  $K_{RA}$  as depicted in Algorithm 4. Then,  $e_P$  loads  $SP_i$  to its EPC to prevent other software components from accessing or falsifying  $SP_i$ .

---

##### Algorithm 4 Protocol for provisioning $SP_i$

---

- 1:  $V_T \leftrightarrow e_P$ : Perform remote attestation  $\rightarrow K_{RA}$
  - 2:  $V_T$ :  $E_i \rightarrow E(SP_i)_{K_{RA}}$
  - 3:  $V_T$ :  $H_i \rightarrow H(E_i)$
  - 4:  $V_T \rightarrow e_P$ :  $E_i, H_i$
  - 5:  $e_P$ : Check  $H_i$  and decrypt  $E_i \rightarrow SP_i$
  - 6:  $e_P$ : Load  $SP_i$
- 

$SP_i$  is a chain of rules that define how SPP behaves on each packet such as PROTECT, BYPASS, or DISCARD. When a packet arrives at SPP, SPP invokes  $e_P$  to find the rule that matches the packet by traversing  $SP_i$  in order. Thus, if  $e_P$  cannot find the matched rule for the packet at a specific trial,  $e_P$  examines the next rules in turn at the next trials. If  $e_P$  finds the matched rule, SPP performs the action indicated by  $e_P$  without traversing the next rules further. Accordingly, the integrity of the rules and their order information is essential for SPP to process packets as intended by  $T$ .

$SP_i$  consists of tuples, and each tuple composes of a rule ( $r_n$ ) and their associated order ( $o_n$ ) where  $n$  is an index of each tuple.  $r_n$  includes IP protocol version (ipv4 or ipv6), direction (in or out), action (bypass, protect, or discard), priority(1,2,3,...), source and destination addresses, source and destination ports (0 – 65535), and protocol (0 – 255). The tuple also contains the SPI(spi) if the action defined in the tuple is protect.  $o_n$  contains the order number (1,2,3,...) of  $r_n$  that SPP examines each packet against  $SP_i$ . The lowest number of  $o_n$  means the last  $r_n$  that SPP examines.

##### Packet processing enclave

$e_P$  is an enclave that provides SPP with the packet processing functions based on SGX SSL [97]. When receiving  $SP_i$  from  $V_T$  via a secure channel between  $V_T$  and  $e_P$ ,  $e_P$  loads  $SP_i$  inside the enclave. After  $V_C$  establishes  $SA_i$  for SPP,  $e_P$  also receives  $SA_i$  via another secure channel between  $e_K$  and  $e_P$ .

and loads  $SA_i$  inside the enclave. SPP supports the protection on  $SA_i$  and  $SP_i$  because  $e_P$  receives them via the secure channels resulted from remote attestation and processes the packets inside the enclave. This isolated execution of packet processing prevents  $Adv.$  from accessing  $SA_i$  and  $SP_i$  illegally.

When receiving a packet, SPP first invokes  $e_P$  to retrieve each entry of  $SP_i$  against the packet. If  $e_P$  finds the matched one, SPP processes the packet through  $e_P$  (PROTECT) or depending on the response from  $e_P$  (BYPASS, DISCARD). If the packet is the target of BYPASS or DISCARD, SPP simply forwards the packet to the destination or drops the packet. In the case of PROTECT,  $e_P$  returns the SPI to SPP for performing the ESP processing on the packet. Using the SPI received from  $e_P$ , SPP invokes  $C_P$  to encrypt (outbound) or decrypt (inbound) the packet. Then,  $C_P$  calls  $e_P$  in turn, to encrypt or decrypt the packet.

SPP presents an optimized invocation between  $C_P$  and  $e_P$ . The existing cryptodev based on OpenSSL [98] processes each packet as follows repeatedly; feeding Initialization Vector (IV) to the encryption or decryption context, encrypting or decrypting the packet, authenticating the packet, finalizing encryption or decryption, and finalizing authentication. If SPP applies this approach in a straightforward way, the throughput decreases drastically because of the frequent enclave transitions. To overcome this inefficiency, SPP realizes the optimized invocation to reduce the number of enclave transitions. First,  $C_P$  executes the encryption/decryption of packets along with IVs at one invocation of  $e_P$ . Second,  $C_P$  does not call  $e_P$  for finalizing the encryption/decryption if the packets are aligned to the block size of the underlying cryptographic algorithm. Third,  $C_P$  performs the authentication of the packets at one invocation of  $e_P$ .

#### Processing packets using $e_P$

SPP processes inbound and outbound packets using  $e_P$  as depicted in Figure 4.3.

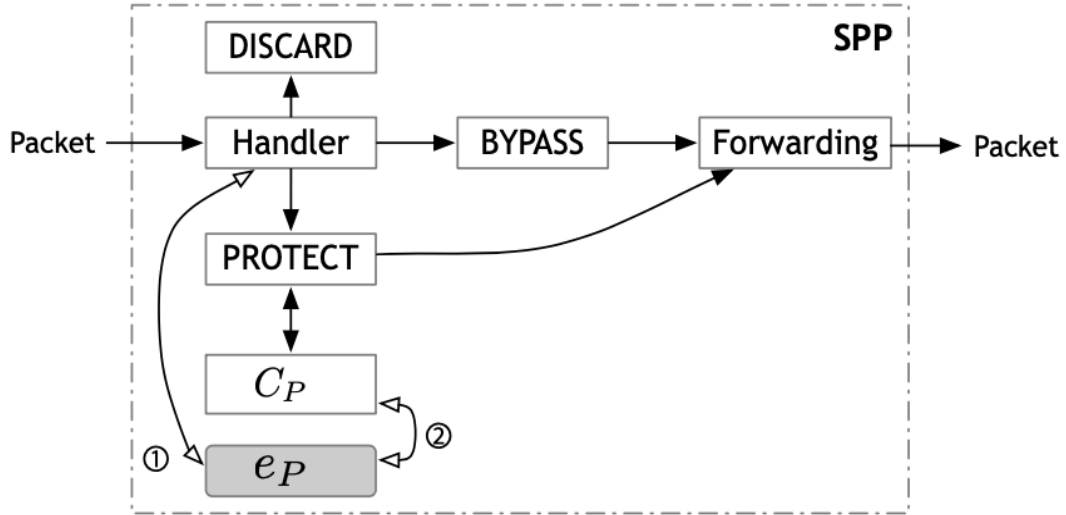


Figure 4.3: Packet processing using  $e_P$ ; packet flows ( $\rightarrow$ ) and invocations of  $e_P$  ( $\rightarrow\rhd$ )

When receiving an outbound packet, SPP calls  $e_P$  to find matched rule from  $SP_i$  (①). Depending on the indication from  $e_P$ , SPP processes the packet: BYPASS, DISCARD or PROTECT. If the packet is the target of PROTECT, SPP invokes  $e_P$  via  $C_P$  to encrypt and integrity protect the packet using the SPI from  $e_P$  (②). Then, SPP forwards the processed packet to be routed if the packet is the target of BYPASS or PROTECT.



The inbound packet processing is quite different from the outbound packet processing. When receiving an inbound packet, SPP examines if the packet needs SA lookups by checking the presence of the SPI in the packet's header. If the packet includes the SPI, SPP invokes  $e_P$  via  $C_P$  to integrity check and decrypt the packet using  $SA_i$  indexed by the SPI (②). After SPP completes the integrity check and decryption on the packet,  $e_P$  checks if the packet matches with  $SA_i$  (②). If the verification against  $SA_i$  fails, SPP drops the decrypted packet. If the packet does not include the SPI, SPP calls  $e_P$  to find matched rule for the packet (①). Using the indication from  $e_P$ , SPP processes the packet: BYPASS or DISCARD. Then, SPP also forwards the processed packet to be routed if the packet is the target of BYPASS or PROTECT.

---

**Algorithm 5** Pseudo-code for outbound packet processing

---

```

1:  $r \leftarrow \text{search-rule}(p)$  via  $e_P$ 
2: if  $r$  indicates discard then  $\text{drop}(p)$ 
3: else if  $r$  indicates bypass then  $\text{enqueue}(p)$ 
4: else if  $r$  indicates protect then
5:   if spi found then  $\text{enqueue}(p, \text{spi})$ 
6:   else  $\text{drop}(p)$ 
7: else  $\text{drop}(p)$ 
8:  $p \leftarrow \text{dequeue}()$ 
9: if  $p$  is enqueued with spi then
10:   $p \leftarrow \text{process-outbound}(p, \text{spi})$  via  $C_P$  ( $e_P$ )
11: return  $\text{route}(p)$ 

```

---

Algorithm 5 shows the pseudo-code of processing the outbound packets using  $e_P$ . SPP processes an outbound packet ( $p$ ) using  $e_P$  as follows:

1. SPP calls  $e_P$  to search the rule ( $r$ ) matched to  $p$ .
2. If  $e_P$  finds the matched rule,  $e_P$  returns the result (**bypass**, **discard** or **protect/spi**) to SPP.
3. SPP drops  $p$  if  $e_P$  indicates **discard** or  $e_P$  cannot find any matched rule.
4. SPP simply enqueues  $p$  if  $p$  is the target of **bypass**.
5. SPP enqueues  $p$  with **spi** if  $p$  is the target of **protect**. If no SPI for  $p$  is found, SPP drops  $p$ .
6. SPP dequeues  $p$ .
7. SPP calls  $e_P$  via  $C_P$  to encrypt  $p$  with integrity protection if it is enqueued with **spi**.
8. SPP routes the processed packet (**protect** or **bypass**) to forward  $p$  to the destination.

Algorithm 6 shows the pseudo-code of processing the inbound packets using  $e_P$ . SPP processes an inbound packet ( $p$ ) using  $e_P$  as follows:

1. SPP checks if  $p$  includes **spi** in its header.
2. If  $p$  does not include **spi**, SPP calls  $e_P$  to search the rule ( $r$ ) matched to  $p$ .
3. If  $e_P$  finds the matched rule,  $e_P$  returns the result (**bypass** or **discard**) to SPP.
4. SPP drops  $p$  if  $e_P$  indicates **discard** or  $e_P$  cannot find any matched rule.
5. SPP simply enqueues  $p$  if  $p$  is the target of **bypass**.



---

**Algorithm 6** Pseudo-code for inbound packet processing

---

```
1: if  $p$ 's header includes spi then
2:   if  $SA_i$  for spi found then enqueue( $p$ , spi)
3:   else drop( $p$ )
4: else
5:    $r \leftarrow \text{search-rule}(p)$  via  $e_P$ 
6:   if  $r$  indicates discard then drop( $p$ )
7:   else if  $r$  indicates bypass then enqueue( $p$ )
8:   else drop( $p$ )
9:  $p \leftarrow \text{dequeue}()$ 
10: if  $p$  is enqueued with spi then
11:    $p \leftarrow \text{process-inbound}(p, \text{spi})$  via  $C_P$  ( $e_P$ )
12:   if  $p$  is not valid with  $SA_i$  then drop( $p$ )
13: return route( $p$ )
```

---

6. If  $p$  includes **spi**, SPP enqueues  $p$  with **spi**. If  $e_P$  cannot find any  $SA_i$  that **spi** indexes, SPP drops  $p$ .
7. SPP dequeues  $p$ .
8. SPP invokes  $e_P$  via  $C_P$  to integrity check and decrypt  $p$  if it is enqueued with **spi**.
9.  $e_P$  checks if the processed packet is appropriate by matching the packet against the selectors defined in  $SA_i$ . If this verification fails, SPP drops  $p$ .
10. SPP routes the processed packet (**protect** or **bypass**) to forward  $p$  to the destination.

**Remote verification on security policies.**

SPP provides remote verification on  $SP_i$  to  $T$ . This verification leverages remote attestation to generate the evidence on  $SP_i$  as depicted in Algorithm 7.  $V_T$  sends a nonce ( $N$ ) with a command message ( $msg$ ) to initialize the remote verification. After receiving  $N$  and  $msg$  via SPP,  $e_P$  requests QE to generate *QUOTE* against  $N$ ,  $msg$ , and  $SP_i$ . Then,  $e_P$  replies  $SP_i$  and *QUOTE* to  $V_T$ , and  $V_T$  verifies *QUOTE* using IAS. If the verification succeeds,  $T$  checks if  $SP_i$  is correct or not.

---

**Algorithm 7** Protocol for remote verification of  $SP_i$ 

---

```
1:  $V_T \rightarrow \text{SPP}: N, msg$ 
2:  $e_P$ : Generate QUOTE against  $N$ ,  $msg$ , and  $SP_i$ 
3:  $\text{SPP} \rightarrow V_T: SP_i, QUOTE$ 
4:  $V_T$ : Verify QUOTE and  $SP_i$ 
```

---

## 4.3 Evaluation

### 4.3.1 Analysis

We evaluate SGX-VPN from the perspective of the goals defined in Section 4.1.3. We utilize Scyther [20], one of formal analysis tools, to prove secrecy of cryptographic keys (**G2**) and integrity protection of SPs (**G3**).

Scyther [20] is an automated protocol verification tool that supports the verification, the falsification, and the analysis of security protocols. Because Scyther is based on a pattern refinement algorithm,

this tool can prove the correctness of protocols for an unbounded number of sessions with guaranteed termination and also assist in protocol analysis by providing classes of protocol behavior (or classes of attacks). Due to the performance of Scyther, it is also possible to support multi-protocol analysis. Scyther leverages the `spdl` language to describe a protocol, and this language is based on the operational semantics found in [99]. With these features, Scyther enables to verify if the security claims in the protocol description hold or not; to produce appropriate security claims for a protocol and verify them automatically; to analyze the protocol by executing complete characterization. Characterization means that Scyther analyzes the protocol, and provides a finite representation of all traces that contain an execution of the protocol role. This tool can verify that a certain value is confidential (secrecy) or certain properties (authentication) should hold for the communication partners. Scyther is appropriate for the analysis of SGX-VPN because this tool can prove the secrecy of cryptographic keys including the key for integrity protection on SPs.

SGX-VPN relies on  $e_K$  and  $e_P$  to execute key exchange and packet processing, respectively. SGX-VPN provisions the sensitive data to these enclaves by leveraging secure channels established by remote attestation. With the strong isolation that SGX supports [19] and the securely provisioned data, the enclaves in SGX-VPN perform the tasks in the isolated memory region without revealing any information. Moreover, SGX-VPN provides each tenant with its own dedicated enclaves for cloud VPN. Thus, *Adv.* cannot violate the isolated execution of cloud VPN for each tenant that SGX-VPN supports even in the semi-trusted cloud environment. **(G1)**

```

/*
 * Protocol for algorithm 1
 * Secure provisioning of C and CFG
 */

// The protocol description
usertype DataForProvisioning;
hashfunction H1;

protocol algorithm1(I, R)
{
    # V_T
    role I
    {
        fresh CCFG: DataForProvisioning;
        send_1(I, R, {CCFG}_k(I,R), H1({CCFG}_k(I,R)));
        claim(I, Secret, CCFG);
    }

    # V_C(eK)
    role R
    {
        var CCFG: DataForProvisioning;
        recv_1(I, R, {CCFG}_k(I,R), H1({CCFG}_k(I,R)));
        claim(R, Secret, CCFG);
    }
}

```

Figure 4.4: Scyther script for Algorithm 1

$V_T$  provisions  $C$  to  $e_K$  via a secure channel established as a result of remote attestation.  $e_K$  generates  $EK$  inside the enclave, and this ephemeral key does not leave the enclave.  $e_K$  derives  $SK$  from  $EK$  and sends  $SA_i$  including  $SK$  to  $e_P$  through a secure channel established by local and remote attestation.  $e_P$  processes packets using  $SK$  inside the enclave, and this session key never leaves the enclave. Thus, *Adv.* cannot extort  $C$ ,  $EK$ , and  $SK$  from the secure channels because only legitimate

Claim			Status	Comments
algorithm1	I	algorithm1,I1	Secret CCFG	Ok Verified No attacks.
	R	algorithm1,R1	Secret CCFG	Ok Verified No attacks.

Done.

KAIST

Figure 4.5: Analysis result of Algorithm 1

enclaves and SGX-enabled processors can perform attestation successfully. Furthermore, *Adv.* cannot access the isolated memory region where the enclaves execute and these keys reside. We perform formal analysis on Algorithm 1, 2, 3 using Scyther to prove the secrecy of these keys. Figure 4.4, 4.6, and 4.8 show the Scyther scripts for this analysis. Figure 4.5, 4.7, and 4.9 show the analysis results from Scyther. (G2)

$V_T$  provisions  $SP_i$  to  $e_P$  via a secure channel established by remote attestation. Using remote attestation, SPP provides  $T$  with an on-demand verification that checks if the intended  $SP_i$  is running in  $e_P$  without any illegal modification. Thus, *Adv.* can neither alter  $SP_i$  transferring to  $e_P$  nor generate the valid QUOTE because only legitimate enclaves and SGX-enabled processors can perform attestation successfully. Moreover, *Adv.* cannot access the isolated memory region where  $SP_i$  resides. We also perform formal analysis on Algorithm 4, 7 using Scyther to prove the secrecy of the provisioned  $SP_i$  and the key for generating the valid QUOTE. Figure 4.10 and 4.12 show the Scyther scripts for this analysis. Figure 4.11, 4.13, and 4.9 show the analysis results from Scyther. (G3)

*Adv.* cannot sniff the packets transmitted via SGX-VPN because SPP encrypts and integrity-protects all packets between  $V_T$  and  $V_C$  and within the VPC network using  $e_P$  via  $C_P$ . To this end, SGX-VPN presents an architecture that adds SPP to each VM in the VPC network and manages the security associations efficiently. (G4)

### 4.3.2 Comparison

Table 4.3 shows the comparison between SGX-VPN and the existing approaches analogous to SGX-VPN [2, 3] against the goals defined in Section 4.1.3. Protego [2] and sVPN [3] provide each tenant with the hypervisor-based shared VPN services (weak isolation [18]). While SGX-VPN leverages enclaves that guarantee the strong isolation [19] to support the isolated execution of VPN services for each tenant. In the semi-trusted cloud environment, *Adv.* can extort cryptographic keys (e.g., session keys) of Protego and sVPN because *Adv.* can compromise the hypervisor. In SGX-VPN, the cryptographic keys reside in the isolated memory region where *Adv.* cannot access, and the session keys transfer to other entities only if the entities have been attested to be trustworthy by remote attestation. None of the existing approaches provides the tenants with remote verification of the SPs running in cloud VPN. Like the existing approaches, SGX-VPN also encrypts and integrity protects all packets within the VPC networks. We mainly focus on the evaluation of the security aspects defined in the goals of Section 4.1.3. This evaluation is useful for the government or the military because cloud VPNs for these organizations can consider the security aspects as being more important than the performance.

```

/*
 * Protocol for algorithm 2
 * IKEv2 & Send IPsec SA for V_C securely
 */
hashfunction prf, KDF;
hashfunction H1;
hashfunction g, h;
hashfunction MAC;

usertype Number, SecurityAssociation, TrafficSelector, IPsecSA;
const 0: Number;
const SA1, SA2, SA3: SecurityAssociation;
const TS1, TSr: TrafficSelector;

protocol algorithm2(I, R, S)
{
  # V_T
  role I {
    fresh i, Ni, SPIi: Nonce;
    var Nr, SPIr: Nonce;
    var Gr: Ticket;

    send_1( I, R, SPIi, 0, SA1, g(i), Ni );
    recv_2( R, I, (SPIi, SPIr), SA1, Gr, Nr );

    send_!3( I, R, (SPIi, SPIr), {I, R, MAC(k(I, R), SPIi, 0, SA1, g(i), Ni, Nr, prf(KDF(Ni, Nr, h(Gr, i), SPIi, SPIr), I))), SA2, TS1, TSr}KDF(Ni, Nr, h(Gr, i), SPIi, SPIr) );
    recv_!4( R, I, (SPIi, SPIr), {R, MAC(k(I, R), SPIi, SPIr, SA1, Gr, Nr, Ni, prf(KDF(Ni, Nr, h(Gr, i), SPIi, SPIr), R))), SA2, TS1, TSr}KDF(Ni, Nr, h(Gr, i), SPIi, SPIr) );
    claim(I, SKR, KDF(Ni, Nr, h(Gr, i), SPIi, SPIr));
  }

  # V_C(eK)
  role R {
    fresh r, Nr, SPIr: Nonce;
    fresh SA4: IPsecSA;
    var Ni, SPIi: Nonce;
    var Gi: Ticket;

    recv_1( I, R, SPIi, 0, SA1, Gi, Ni );
    send_2( R, I, (SPIi, SPIr), SA1, g(r), Nr );

    recv_!3( I, R, (SPIi, SPIr), {I, R, MAC(k(R, I), SPIi, 0, SA1, Gi, Ni, Nr, prf(KDF(Ni, Nr, h(Gi, r), SPIi, SPIr), I))), SA2, TS1, TSr}KDF(Ni, Nr, h(Gi, r), SPIi, SPIr) );
    send_!4( R, I, (SPIi, SPIr), {R, MAC(k(R, I), SPIi, SPIr, SA1, g(r), Nr, Ni, prf(KDF(Ni, Nr, h(Gi, r), SPIi, SPIr), R))), SA2, TS1, TSr}KDF(Ni, Nr, h(Gi, r), SPIi, SPIr) );
    claim(R, SKR, KDF(Ni, Nr, h(Gi, r), SPIi, SPIr));

    send_5(R, S, {SA4}k(R, S), H1({SA4}k(R, S)));
    claim(R, Secret, SA4);
  }

  # V_C(eP)
  role S{
    var SA4: IPsecSA;
    recv_5(R, S, {SA4}k(R, S), H1({SA4}k(R, S)));
    claim(S, Secret, SA4);
  }
}

```

Figure 4.6: Scyther script for Algorithm 2

Claim				Status	Comments
algorithm2	I	algorithm2.I1	Secret KDF(Ni,Nr,h(Gr,i),SPi,SPir)	OK	No attacks within bounds.
	R	algorithm2.R1	Secret KDF(Ni,Nr,h(Gi,r),SPi,SPir)	OK	No attacks within bounds.
		algorithm2.R2	Secret SA4	OK	No attacks within bounds.
	S	algorithm2.S1	Secret SA4	OK	No attacks within bounds.

Done.

Figure 4.7: Analysis result of Algorithm 2

```

/*
 * Protocol for algorithm 3
 * Send IPsec SA for VMs securely
 */
hashfunction prf, KDF;
hashfunction H1;
hashfunction g, h;

usertype SecurityAssociation, IPsecSA;
const SA1, SA2, SA3: SecurityAssociation;

protocol algorithm3(I, R, S)
{
  # V_T
  role I {
    fresh i, Ni: Nonce;
    var Nr: Nonce;
    var Gr: Ticket;

    send_!1( I, R, {SA3, Ni, g(i)}k(I,R) );
    recv_!2( R, I, {SA3, Nr, Gr}k(I,R) );

    claim( I, SKR, KDF(k(I,R),h(Gr,i),Ni,Nr) );
  }

  # V_C(eK)
  role R {
    fresh r, Nr: Nonce;
    fresh SA4: IPsecSA;
    var Ni: Nonce;
    var Gi: Ticket;

    recv_!1( I, R, {SA3, Ni, Gi}k(R,I) );
    send_!2( R, I, {SA3, Nr, g(r)}k(R,I) );

    claim( R, SKR, KDF(k(R,I),h(Gi,r),Ni,Nr) );

    send_3(R, S, {SA4}k(R,S), H1({SA4}k(R,S)));
    claim(R, Secret, SA4);
  }

  # VMs(eP)
  role S{
    var SA4: IPsecSA;
    recv_3(R, S, {SA4}k(R,S), H1({SA4}k(R,S)));
    claim(S, Secret, SA4);
  }
}

```

Figure 4.8: Scyther script for Algorithm 3





Scyther results : verify

Claim				Status	Comments
algorithm3	I	algorithm3,I1	Secret KDFik(I,R),h(Gr,i),Ni,Nr)	Ok	No attacks within bounds.
	R	algorithm3,R1	Secret KDFik(R,I),h(Gr,r),Ni,Nr)	Ok	No attacks within bounds.
		algorithm3,R2	Secret SA4	Ok	No attacks within bounds.
	S	algorithm3,S1	Secret SA4	Ok	No attacks within bounds.

Done.

Figure 4.9: Analysis result of Algorithm 3

```

/*
 * Protocol for algorithm 4
 * Secure provisioning of SP
 */

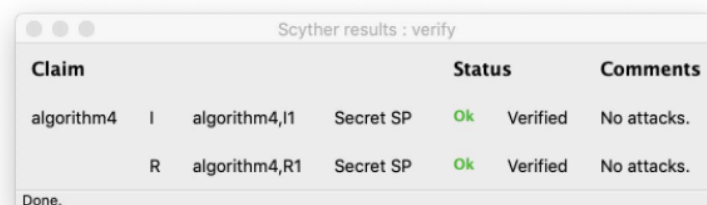
// The protocol description
usertype DataForProvisioning;
hashfunction H1;

protocol algorithm4(I, R)
{
    # V_T
    role I
    {
        fresh SP: DataForProvisioning;
        send_1(I, R, {SP}k(I,R), H1({SP}k(I,R)));
        claim(I, Secret, SP);
    }

    # eP (V_C/VMs)
    role R
    {
        var SP: DataForProvisioning;
        recv_1(I, R, {SP}k(I,R), H1({SP}k(I,R)));
        claim(R, Secret, SP);
    }
}

```

Figure 4.10: Scyther script for Algorithm 4



Scyther results : verify

Claim				Status	Comments
algorithm4	I	algorithm4,I1	Secret SP	Ok	Verified No attacks.
	R	algorithm4,R1	Secret SP	Ok	Verified No attacks.

Done.

Figure 4.11: Analysis result of Algorithm 4

```

/*
 * Protocol for algorithm 5
 * Remote verification of SP
 */

// The protocol description
usertype SecurityPolicy, QuoteKey, VerificationCommand;
usertype QuoteResult;
hashfunction H1;
const msg: VerificationCommand;
const PK: Function;
secret QK: Function;
inversekeys(PK, QK);

protocol algorithm5(I, R, IAS)
{
    macro Quote = H1(N, SP, QK);

    # V_T
    role I
    {
        fresh N: Nonce;
        var SP: SecurityPolicy;
        var Result: QuoteResult;
        send_1(I, R, N, msg);
        recv_2(R, I, SP, Quote);
        send_3(I, IAS, Quote);
        recv_4(IAS, I, Result);
        claim(I, Secret, QK);
    }

    # eP (V_C/VMs)
    role R
    {
        var N: Nonce;
        fresh SP: SecurityPolicy;
        recv_1(I, R, N, msg);
        send_2(R, I, SP, Quote);
        claim(R, Secret, QK);
    }

    # Attestation Service
    role IAS
    {
        var SP: SecurityPolicy;
        var N: Nonce;
        fresh Result: QuoteResult;
        recv_3(I, IAS, Quote);
        send_4(IAS, I, Result);
        claim(IAS, Secret, QK);
    }
}

```

Figure 4.12: Scyther script for Algorithm 7

Table 4.3: Comparison with the existing approaches; ○ - support, × - not support, N/A - Not Available.

	SGX-VPN	sVPN [3]	Protego [2]
<b>G1</b>	Strong	Weak	Weak
<b>G2</b>	○	×	×
<b>G3</b>	○	N/A	N/A
<b>G4</b>	○	○	○

G1 - Which isolation supports?

G2 - Does secrecy of cryptographic keys support?

G3 - Does integrity protection on SPs support?

G4 - Does encryption on all packets support?

Claim	Status	Comments
algorithm5 I algorithm5,I1 Secret QK	Ok Verified	No attacks.
algorithm5 R algorithm5,R1 Secret QK	Ok Verified	No attacks.
algorithm5 IAS algorithm5,IAS1 Secret QK	Ok Verified	No attacks.

Done.

Figure 4.13: Analysis result of Algorithm 7

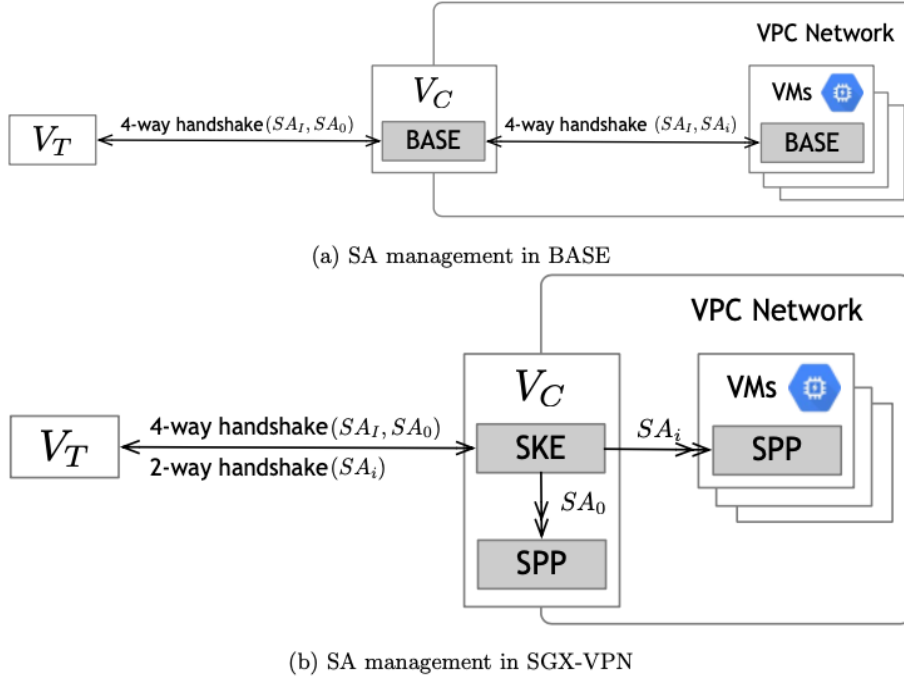


Figure 4.14: SA management in BASE and SGX-VPN

### 4.3.3 Efficiency in SA management

SGX-VPN presents an architecture that has benefits in the number of SAs and key exchange messages for protecting packets within the VPC network as shown in Table 4.4 and Figure 4.14. In the existing cloud VPN (called BASE), each VM should manage additional  $N$   $SA_I$  and  $N$   $SA_i$ . However, SGX-VPN reduces this addition only to  $N$  because SGX-VPN uses the existing  $SA_I$  of  $V_C$  to manage  $N$   $SA_i$  for each VM. Moreover, each VM in BASE should process additional  $4N$  key exchange messages because the existing key exchange protocol [16] necessitates the 4-way handshake protocol to establish each  $SA_I$  and  $SA_i$ . Meanwhile, SKE in SGX-VPN processes only  $2N$  additional messages because SGX-VPN extends the existing 2-way handshake protocol [16] to negotiate additional  $SA_i$  using the existing  $SA_I$ . For the SA management, SGX-VPN leverages SKE running in  $V_C$  that the CSPs operate reliably. However, BASE utilizes each VM for the SA management, and thus tenants should guarantee the reliable operations of the VMs.

Table 4.4: Additional SAs and key exchange messages for packets within the VPC network;  $N$  is a number of VMs.

	BASE	SGX-VPN
$SA_I$	$N$	0
$SA_i$	$N$	$N$
key exchange messages	$4N$	$2N$
SA Management entity	each VM	SKE in $V_C$

#### 4.3.4 Performance

The utilization of SGX into cloud VPN introduces inevitable performance penalty in processing packets. SPP invokes `ecalls` to execute features implemented in  $e_P$ . These `ecalls` usually consumes 8000 CPU cycles, and this overhead is 50 times more expensive than a system call [100].

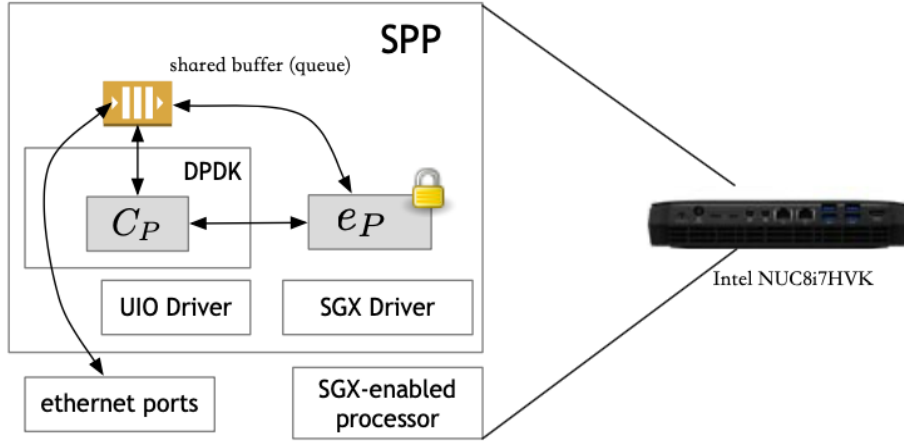


Figure 4.15: A prototype of SPP

#### Implementation

We have implemented a prototype of SPP in an Intel NUC8i7HVK (Core i7-8809G 3.10GHz quad core CPU) machine running Ubuntu 16.04 LTS (64-bit) as shown in Figure 4.15. We implement  $C_P$  that is compatible with DPDK 18.11 [21] and leverages SGX SSL [97]. We use the SGX Software Development Kit (SDK) for Linux [101] to implement  $e_P$ .

The prototype utilizes a shared buffer, queue of DPDK framework, between  $e_P$  and  $C_P$  to process packets. Receiving the memory addresses of the packets from  $C_P$ ,  $e_P$  processes the packets without copying the packets into its own EPC pages. This approach enhances the throughput of packet processing and is widely deployed in previous work [88]. When SPP receives the packets from ethernet ports, SPP enqueues the packets into the shared buffer. Then,  $C_P$  processes the packets via  $e_P$  and SPP forwards the processed packets to the destination.

First, we implement  $C_P$  to invoke `ecalls` that  $e_P$  provides for processing packets in a straightforward way as OpenSSL-based cryptODEV in DPDK supports (called SPP V1). That is, to encrypt a single outbound packet or decrypt a single inbound packet, SPP  $C_P$  invokes three `ecalls` to call `C.En/DecryptInit.ex()` for feeding an IV to the existing en/decryption context, `C.En/DecryptUpdate` for performing encryption or decryption, and `C.En/DecryptFinal.ex()` for finalizing encryption or decryption. To generate or verify the authentication data of the packet,  $C_P$  also invokes three `ecalls` to

Table 4.5: Packet processing that  $e_P$  supports for SPP V1

Operation	ecall	Description
Encryption	<code>sgx_packet_encrypt_init()</code>	Feed an IV to the existing encryption context inside $e_P$
	<code>sgx_packet_encrypt_update()</code>	Perform packet encryption inside $e_P$
	<code>sgx_packet_encrypt_final()</code>	Finalize packet encryption inside $e_P$
Decryption	<code>sgx_packet_decrypt_init()</code>	Feed an IV to the existing decryption context inside $e_P$
	<code>sgx_packet_decrypt_update()</code>	Perform packet decryption inside $e_P$
	<code>sgx_packet_decrypt_final()</code>	Finalize packet decryption inside $e_P$
Authentication	<code>sgx_packet_auth_update()</code>	Perform packet authentication inside $e_P$
	<code>sgx_packet_auth_final()</code>	Finalize packet authentication inside $e_P$
	<code>sgx_packet_auth_clear()</code>	Clear the authentication context inside $e_P$

call `C_HMAC_Update()` for performing authentication, `C_HMAC_Final()` for finalizing authentication, and `C_HMAC_Init_ex()` for clearing the used authentication context. For this implementation,  $e_P$  implements `ecalls` as depicted in Table 4.5.

As an initial benchmark, we measure the throughput of SPP V1 as shown in Table 4.6. We utilize AES128-CBC HMAC-SHA1, which is one of secure encrypt-then-MAC cipher suites and a widely used cipher suite in benchmarking DPDK-based IPsec gateway, for this measurement. Regarding of the ESP tunnel mode that SGX-VPN supports, encryption only and MAC-then-encrypt configurations are proven to be insecure [102, 103]. The throughput of SPP V1 shows 470.70Mbps against 986.3Mbps, and thus 52.28% packet loss happens. The number of enclave transitions per packet causes this deterioration because the only difference is the utilization of an enclave and enclave transitions are expensive [104].

Table 4.6: Throughput of SPP V1

Packet size(byte)	Throughput(Mbps)
64	29.26
72	32.10
128	51.92
256	93.11
512	175.98
768	252.73
1024	336.44
1280	409.71
1420	470.70

Thus, we optimize SPP V1 to minimize the number of enclave transitions for processing packets using  $e_P$  (called SPP V2). For this optimized implementation,  $e_P$  implements `ecalls` as depicted in Table 4.7. To this end,  $e_P$  implements a function to encrypt or decrypt packets with feeding IVs to the existing cryptographic contexts inside  $e_P$  at one `ecall`. Moreover, according to RFC 4303 [26], ESP packets include the padding field to ensure that the original packets to be encrypted are a multiple of the algorithm’s block size. Thus, SPP adds padding fields to outbound packets, and  $C_P$  adds a routine to check if the packet size is aligned to the block size of the underlying block algorithm. If the packet size is aligned,  $C_P$  does not invoke `ecall` to finalize encryption or decryption.  $e_P$  also implements a function to perform the packet authentication at one `ecall`. This merging mechanism reduces the number of `ecalls`



Table 4.7: Packet processing that  $e_P$  supports for SPP V2

Operation	ecall	Description
Encryption	<code>sgx_packet_fast_encrypt_update()</code>	Perform packet encryption while feeding an IV to the existing encryption context inside $e_P$
	<code>sgx_packet_encrypt_final()</code>	Finalize packet encryption inside $e_P$
Decryption	<code>sgx_packet_fast_decrypt_update()</code>	Perform packet decryption while feeding an IV to the existing decryption context inside $e_P$
	<code>sgx_packet_decrypt_final()</code>	Finalize packet decryption inside $e_P$
Authentication	<code>sgx_packet_fast_auth()</code>	Perform and finalize packet authentication with clearing the authentication context inside $e_P$

from six to two at the most. Consequently, the prototype executes only two `ecalls` to en/decrypt and authenticate a single packet.

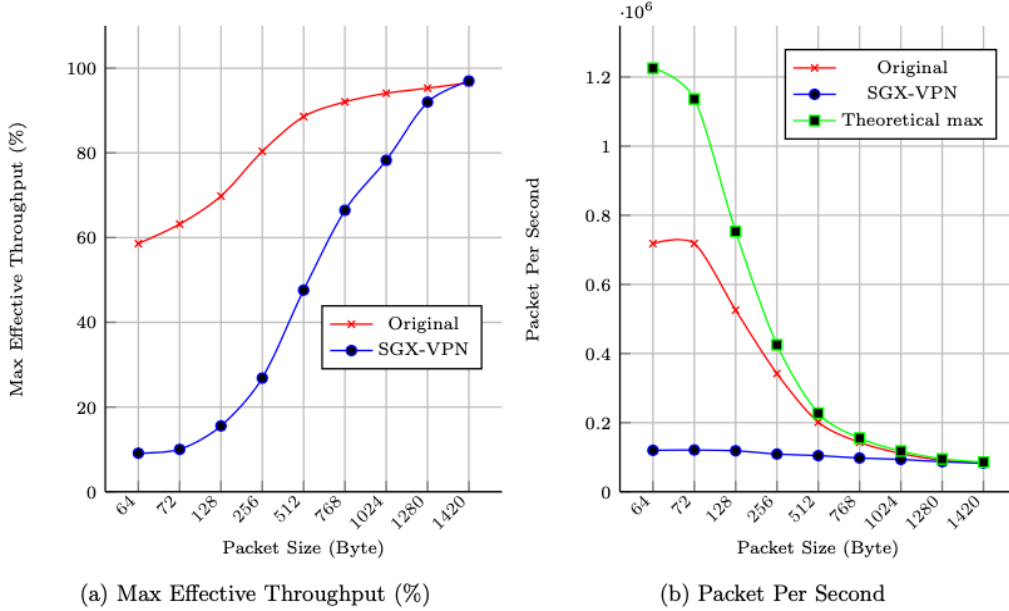


Figure 4.16: Comparison of Throughput and Packet Per Second (AES128-CBC HMAC-SHA256)

### Throughput and Latency

We measure the throughput and latency of SPP in processing packets via  $e_P$ . For this measurement, we leverage SPP V2 (called SGX-VPN), follow a standard RFC 2544 [105], and leverage Agilent N2X, a network tester. For the comparison, we leverage the existing IPsec implementation that uses the OpenSSL-based cryptodev (called Original).

Figure 4.16 and 4.17 show the maximum effective throughput of SGX-VPN, which is a ratio of the maximum theoretic rate to the actual rate when the packet loss does not happen. SGX-VPN shows high throughput (89%, 95%, and 96%), which are almost identical to those of Original when the packet size is more than 1024 byte. The average latencies of SGX-VPN for this packet size are also similar to those of Original as shown in Table 4.8.

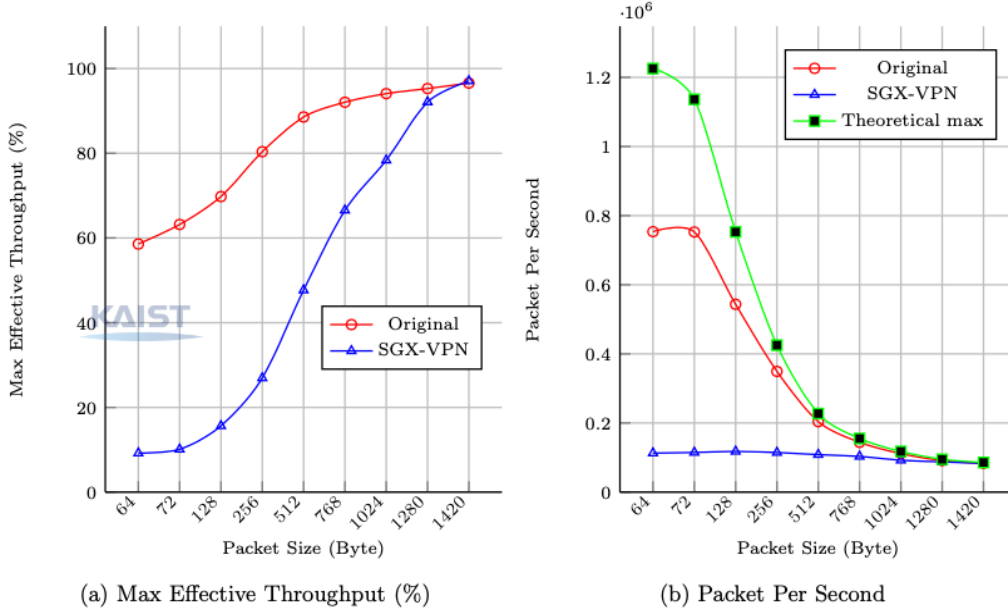


Figure 4.17: Comparison of Throughput and Packet Per Second (AES128-CTR HMAC-SHA256)

However, as the packet size decreases, SGX-VPN shows the lower throughput than those of Original as depicted in Figure 4.16 and 4.17. The rationale behind this decrement is that the smaller packets inevitably increase the number of enclave transitions. For the given input traffic, it is certain that the number of packets that SGX-VPN should process per second increases as the packet size decreases. Nevertheless, the overheads of the enclave transitions prevent SGX-VPN from processing more packets, and thus SGX-VPN shows the lower throughput than those of Original. Figure 4.16 and 4.17 also show the number of packets per second (PPS) that SGX-VPN and Original process. For reference, Figure 4.16 and 4.17 depict the number of theoretical PPS. SGX-VPN processes almost the even number of PPS regardless of the packet size, whereas Original processes more packets for the smaller packets. The overheads of the enclave transitions also exacerbate the average latencies of SGX-VPN for the packets whose size is below 1024 byte. Thus, the average latencies of SGX-VPN are much higher than those of Original as shown in Table 4.8.

Table 4.8: Comparison of Average Latency( $\mu$ s)

Packet size (Byte)	Original	SGX-VPN
1420	6683.04	6339.01
1280	6153.62	6662.36
1024	5102.91	5512.51
< 1024	196.85~4038.60	2697.83~7840.82

## Chapter 5. eMotion: An SGX Extension for Migrating Enclaves

### 5.1 Problem Definition

#### 5.1.1 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 with each other and access the same VM image of  $V$ .

#### 5.1.2 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 [106] 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 [106].  $Adv.$  can simply discard messages for executing  $P$ ; such a denial-of-service (DoS) attack is out of scope in eMotion.

#### 5.1.3 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 an 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<sub>S</sub>*, *VMM<sub>D</sub>* 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<sub>S</sub>* should be able to evict the contents of the enclave as well as the associated data structures to the untrusted memory securely. *VMM<sub>D</sub>* 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.

## 5.2 Design

### 5.2.1 Overview

eMotion is an SGX extension for VMMs to migrate the running enclave securely. eMotion consists of additional instructions and migration support. Migration Enclave (ME), a new AE, establishes *MMK* between *H<sub>S</sub>* and *H<sub>D</sub>* 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<sub>S</sub>* and *VMM<sub>D</sub>* can migrate *E* using *MMK*.

Using eMotion, we introduce two phases: *key exchange with remote attestation* (*P<sub>1</sub>*) and *secure eviction and loading for migration* (*P<sub>2</sub>*), which compose *P*. ME executes *P<sub>1</sub>* to establish *MMK* via remote attestation and store *MMK* into MKR (Migration Key Register) of the SGX-enabled processor. *VMM<sub>S</sub>* and *VMM<sub>D</sub>* proceed *P<sub>2</sub>* to migrate *E* using *MMK*.

In this section, we explain eMotion by dividing it into two distinct extensions: one for *P<sub>1</sub>* and the other for *P<sub>2</sub>*. We also present diagrams of two phases and an architecture based on eMotion.

### 5.2.2 SGX Extension for Key Exchange with Remote Attestation

To migrate *E* from *H<sub>S</sub>* to *H<sub>D</sub>* securely, *H<sub>S</sub>* and *H<sub>D</sub>* should establish *MMK* first. MKR of the SGX-enabled processor should store the established *MMK* for being used in *P<sub>2</sub>*. Other entities except for the designated enclave and the SGX-enabled processor should not be able to access this key.

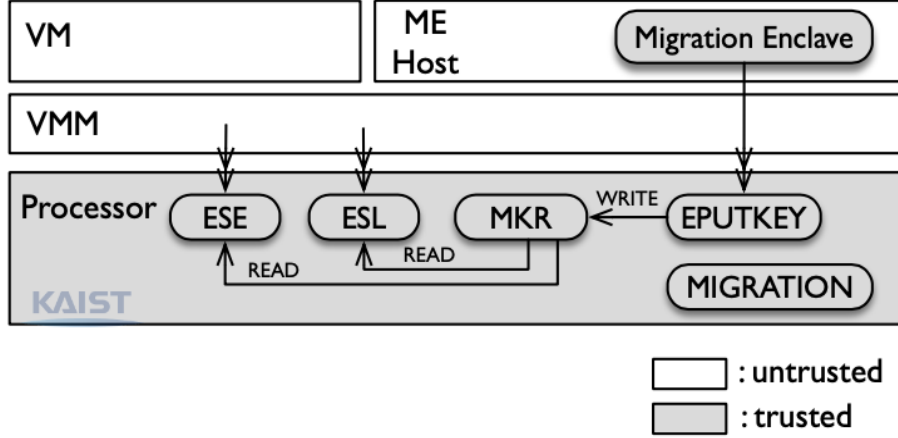


Figure 5.1: eMotion; register read/write ( $\rightarrow$ ) and instruction execution ( $\rightarrow\rightarrow$ )

#### 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 [107] 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 8. ME generates a key exchange message ( $msg$ ). ME calculates the hash value,  $h$ , of  $msg$  using a cryptographic hash function,  $H(\cdot)$ . ME invokes `LocalAttest( $\cdot$ )` to obtain `REPORT` that includes a MAC tag for local attestation. `LocalAttest( $\cdot$ )`, 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 [1]. 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 8 Protocol for manipulating key exchange messages

---

- 1: ME :      Generate key exchange message,  $msg$
  - 2:             $h \leftarrow H(msg)$
  - 3:            `REPORT`  $\leftarrow$  `LocalAttest( $h$ )`
  - 4: ME $\rightarrow$ QE :  $msg$ , `REPORT`
  - 5: QE:        if `REPORT` is valid then Generate `QUOTE` else Abort
  - 6: QE $\rightarrow$ ME : `QUOTE`
  - 7: ME:        return  $msg$ , `QUOTE`
-



ME executes Algorithm 9 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-operated service called Intel Attestation Service (IAS) [96] 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 8. After remote attestation completes successfully, MEs invoke  $Derive(\cdot)$  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 [1, 23, 29, 30, 96] for further details.

---

**Algorithm 9** Protocol for deriving  $MMK$

---

```

1: if  $QUOTE'$  is valid then
2:   if  $msg$  is NULL then Execute Algorithm 8
3:    $MMK \leftarrow Derive(msg, msg')$ 
4: else Abort
5: return  $MMK$ 

```

---

**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.,  $PROVISIONKEY$ ) 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  $MIGRATION$ s 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  $EINITTOKEN$ s from LE among enclaves that attempt to set their  $MIGRATION$ s to true. Without a valid  $EINITTOKEN$ , any enclave cannot be launched in the SGX-enabled processor [30].

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.

Diagram of Key Exchange with Remote Attestation.

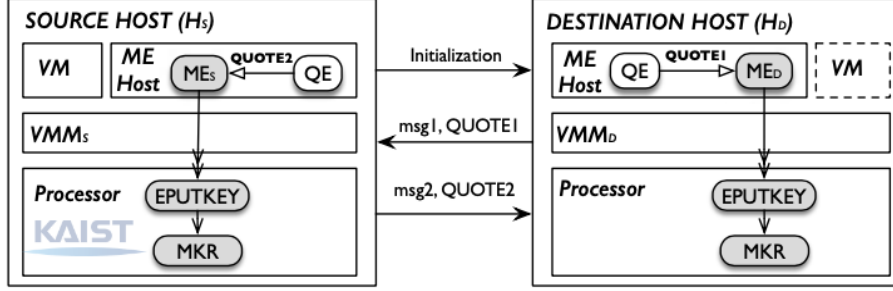


Figure 5.2: Diagram of key exchange with remote attestation; register read/write ( $\rightarrow$ ) and instruction execution ( $\rightarrow$ )

Figure 5.2 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 Figure 5.2 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.

### 5.2.3 SGX Extension for Secure Eviction and Loading for Migration

#### 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 [108] as a key derivation function that the SGX implementation supports [30]. Suppose that  $KDF(\cdot)$  is the key derivation function that ESE and ESL use. Then,  $MK$  and  $IV$  are derived using Equation 5.1.

$$\begin{aligned} MK &= KDF(MMK, C_{MK}) \\ IV &= KDF(MMK, C_{IV}) \end{aligned} \quad (5.1)$$

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 the 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 VMs 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  $i = 1, 2, 3, \dots$ . Then,  $MK$  and  $IV$  for  $VM_i$  are derived using Equation 5.2.



$$\begin{aligned} MK_i &= \text{KDF}(MMK, C_{MK}, VM_i) \\ IV_i &= \text{KDF}(MMK, C_{IV}, VM_i) \end{aligned} \quad (5.2)$$

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 [109], which is used by the EPC paging instructions and supports both confidentiality and integrity. These instructions also take 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  $IV$ .

Figure 5.3 and 5.4 depict the flow charts of ESE and ESL. For the sake of simplicity, the flow charts omit routines used to check the memory alignment.

ESE (Figure 5.3) 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 (Figure 5.4) 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.

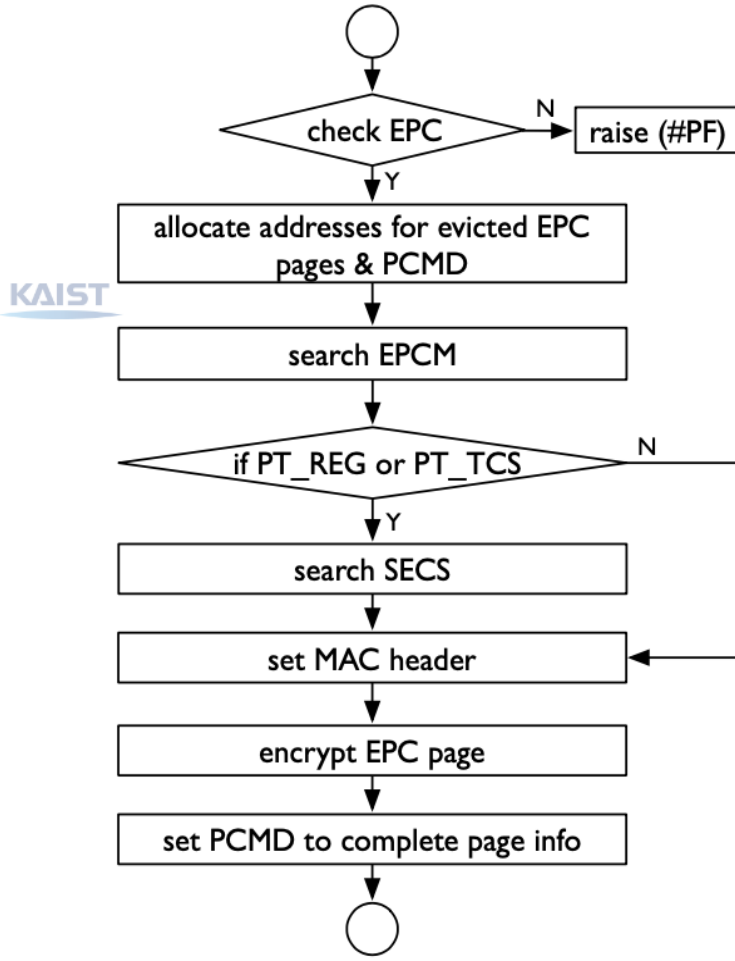


Figure 5.3: Flow chart of ESE

#### 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. Figure 5.5 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

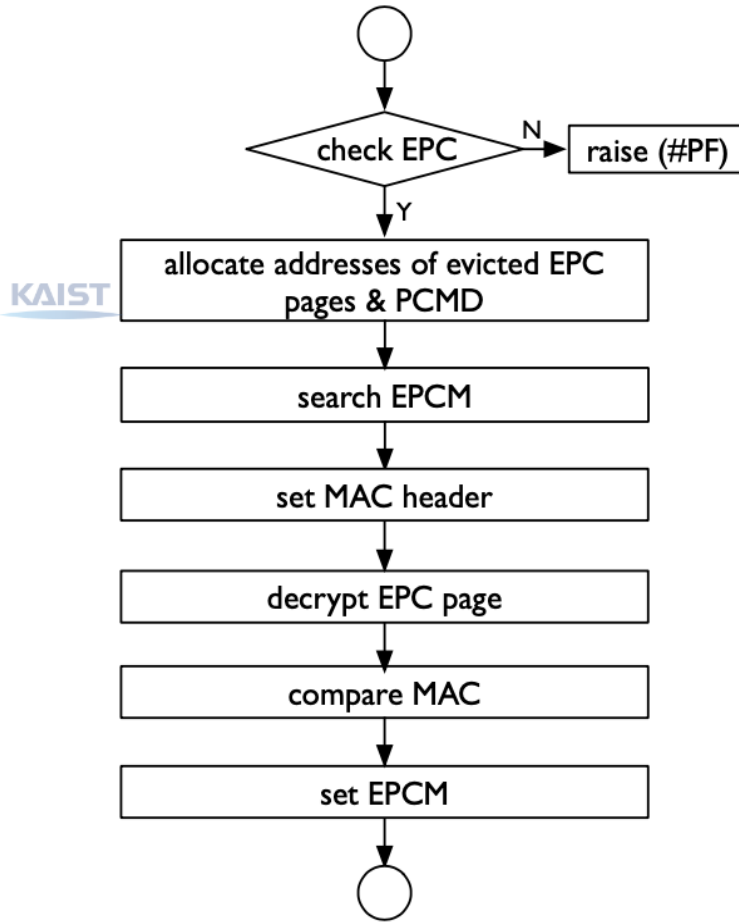


Figure 5.4: Flow chart of ESL

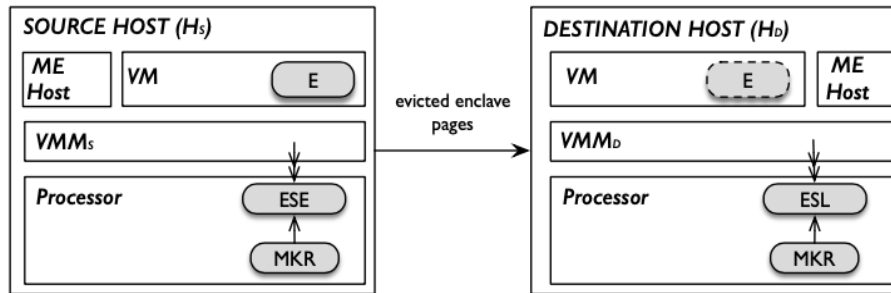


Figure 5.5: Diagram of secure eviction and loading for migration; register read/write ( $\rightarrow$ ), and instruction execution ( $\rightarrow$ )

pages by itself. This mismatch between  $VMM_S$  and  $V$  can be addressed if  $VMM_S$  emulates SGX instructions for  $V$  [110]. 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.



### 5.2.4 eMotion Architecture

Figure 5.6 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  $H_D$ .

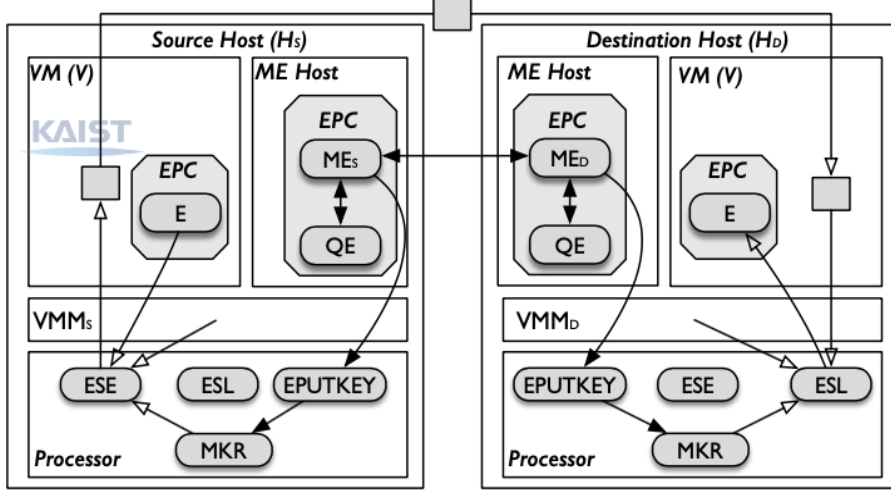


Figure 5.6: An architecture of eMotion; key exchange with remote attestation ( $\rightarrow$ ), and secure eviction and loading for migration ( $\rightarrow$ )

$P_1$  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  $ME_D$  establishes  $MMK$  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  $VMM_D$ . 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  $E$ .

## 5.3 Implementation

We have implemented a prototype of eMotion on top of OpenSGX [9] 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.

### 5.3.1 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 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.

### 5.3.2 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 [107] 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[111] 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 [98] to encrypt and integrity protect the enclave pages based on AES-GCM [109].

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.

### 5.3.3 Implementation Result

In the current prototype, we add a total of 2,286 lines of code to OpenSGX and confirm the operations of eMotion. Figure 5.7 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. (5.7e)
2. When local attestation succeeds,  $ME_D$  responds with QUOTE1 of **msg1** for remote attestation. (5.7f)
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. (5.7b)
4. When local attestation succeeds, the source QE responds with QUOTE2 of **msg2** for remote attestation. (5.7c)
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  $MMK$ . (5.7b and 5.7e)
6. MEs cooperate with  $E$ s in both hosts for executing EPUTKEY to store  $MMK$  to MKR. (5.7a and 5.7d)
7. The **vmm** in  $H_S$  executes ESE to evict the entire enclave pages of  $E$ . (5.7a)
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. (5.7d)

```

sgx@ubuntu:~/opengsx
sgx@ubuntu:~/opengsx$ ./opengsx -l user/demo/lm-target-source.sgx user/demo/lm-target-source.conf source
Hello, this is the source host
sgx target lm: listening to ME on 8024...
received MPK:
0000 46 0a f7 f6 5b 92 ad c6 74 19 7e 54 2c 90 a5 08 F...[...t.-T...
succeed in getting MPK
MPK:
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:~/opengsx$

```

(a)  $E$  in  $H_S$

```

sgx@ubuntu:~/opengsx
sgx@ubuntu:~/opengsx$ ./opengsx user/demo/lm-remote-attest-source-me.sgx user/demo/lm-remote-attest-source-me.conf
Connecting to destination on 8025...
Receiving 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 MPK...
MPK:
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 MPK to target enclave...
key exchange with remote attestation completed!
sgx@ubuntu:~/opengsx$

```

(b)  $ME_S$

```

sgx@ubuntu:~/opengsx
sgx@ubuntu:~/opengsx$ ./opengsx 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:~/opengsx$

```

(c) source QE

```

sgx@ubuntu:~/opengsx
sgx@ubuntu:~/opengsx$ ./opengsx -l user/demo/lm-target-dest.sgx user/demo/lm-target-dest.conf dest
Hello, this is the destination host
sgx target lm: listening to ME on 8023...
received MPK:
0000 46 0a f7 f6 5b 92 ad c6 74 19 7e 54 2c 90 a5 08 F...[...t.-T...
succeed in getting MPK
Listening to source host on 8022...
secure loading for migration: receive & load evicted EPC pages...
MPK:
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 & 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:~/opengsx
sgx@ubuntu:~/opengsx$ ./opengsx 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...
Receiving msg2...
Succeed to verify QUOTE!
Calculating MPK...
MPK:
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 MPK to target enclave...
key exchange with remote attestation completed!
sgx@ubuntu:~/opengsx$

```

(e)  $ME_D$

```

sgx@ubuntu:~/opengsx
sgx@ubuntu:~/opengsx$ ./opengsx user/demo/lm-remote-attest-quote-dest.sgx user/demo/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:~/opengsx$

```

(f) destination QE

Figure 5.7: Implementation result of eMotion in OpenSGX

As indicated by the arrow in Figure 5.7d,  $vmm$  migrates  $E$  using eMotion.

## 5.4 Evaluation

### 5.4.1 Analysis

We evaluate eMotion from the perspective of the goals defined in Section 5.1.

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 enclave 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)

### 5.4.2 Comparison

Table 5.1 shows the comparison between eMotion and the existing migration schemes [7,8,24,25] for SGX enclaves against the goals defined in Section 5.1.3. eMotion and the mechanisms based on the self migration manner [24,25] support the end-to-end protection on migrated enclave pages and the restricted access to the cryptographic key used in enclave migration because only trustworthy entities establish the secure channels directly for secure enclave migration. However, an architecture in Intel's patent [8] cannot guarantee the end-to-end protection on the migrated enclave pages because the trusted server mediates the transportation of the migration capable keys and thus it is difficult to ensure that only the participating hosts can access the migration capable keys. Only eMotion and Intel's patent [7] provides the VMMs with instructions for enclave migration.



Table 5.1: Comparison with the existing migration schemes for SGX enclaves; ○ - support, × - not support.

	eMotion	Gu <i>et al.</i> [24]	Intel’s patents [7, 8]	Alder <i>et al.</i> [25]
<b>G1</b>	○	○	×	○
<b>G2</b>	○	○	×	○
<b>G3</b>	Managed (○)	Self (×)	Managed (○)	Self (×)

G1 - Does end-to-end protection on migrated enclave pages support?

G2 - Can only trustworthy party access *MMK*?

G3 - Which type of enclave migration support?

### 5.4.3 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 5.3. 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.

#### Overhead in key exchange with remote attestation.

Table 5.2 shows the overhead caused by key exchange with remote attestation in ME and QE in terms of the number of instructions. We refer to the model used in [83] to calculate CPU cycles consumed by ME and QE for operating  $P_1$ . Therefore, we assume that each SGX instruction consumes 10K CPU cycles [71], and use 1.8 CPU cycles for each normal instruction [83]. 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.

Table 5.2: The number of instructions in ME and QE during key exchange with remote attestation

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

#### 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 a 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 to 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 5.3 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 Figure 5.3 and 5.4, 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



Table 5.3: Elapsed time for secure eviction and loading for migration

	$VMM_S$	$VMM_D$
elapsed time (ms)	6.69ms	4.31ms

(475 EPC pages) as mentioned in [9]. Figure 5.8 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.

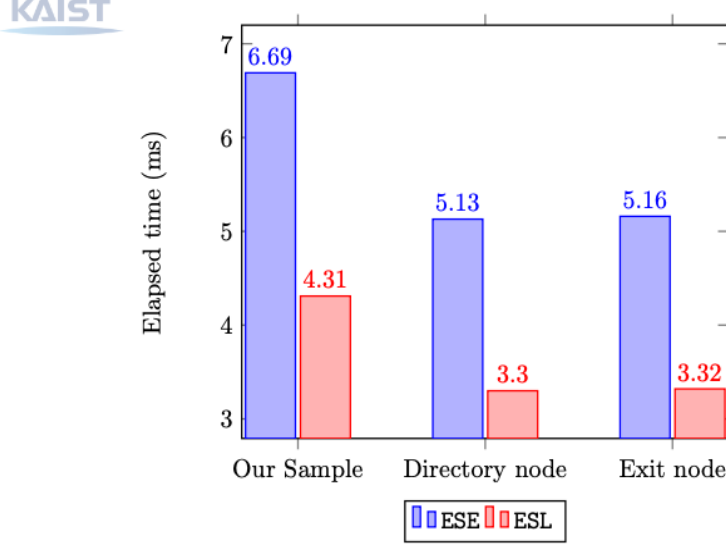


Figure 5.8: Elapsed time for secure eviction and loading for migration on enclaves; Directory node and Exit node are Tor enclaves in [9]

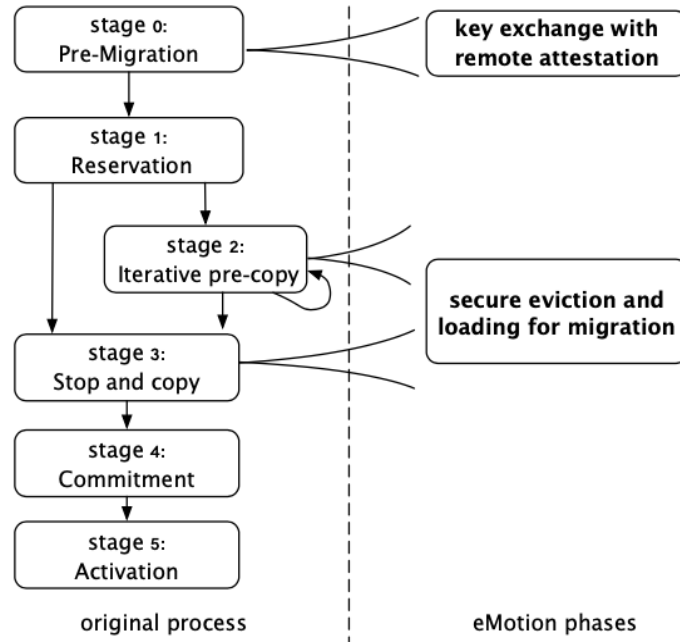


Figure 5.9: Possible deployments of eMotion to pre-copy approach [10]

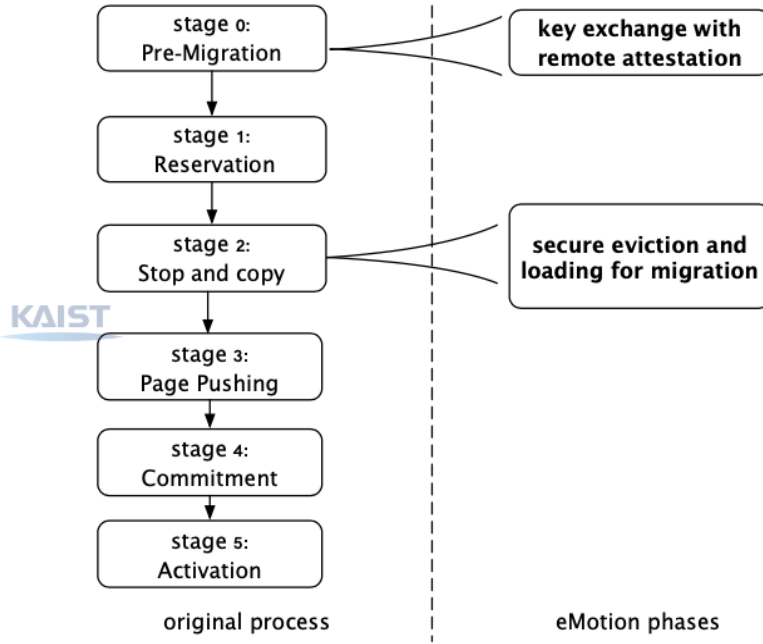


Figure 5.10: Possible deployments of eMotion to post-copy approach [11]

## 5.5 Possible Deployments

The existing live migration of VMs [10, 11] can use eMotion by adding two phases, as depicted in Figure 5.9 and 5.10. 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 operate using the established migration master key when the VMM encounters the EPC pages. Similarly, other live migration protocols [112] can add eMotion to support live migration of SGX-enabled VMs.

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 [113, 114] can launch before enclave migration to verify if the genuine VMM launched and is running on the source host. Moreover, users can utilize the existing detection mechanism [106] to detect the rollback and forking attacks that the active adversaries can perform.

## Chapter 6. Concluding Remark

### 6.1 Conclusion

In this dissertation, we present SGX-VPN that leverages enclaves and attestation to support the security-enhanced cloud VPN for each tenant in the semi-trusted cloud environment. With the enclaves, SGX-VPN provides each tenant with the strong isolation of key exchange and packet processing. SGX-VPN also provides each tenant with an on-demand functionality to verify the integrity of the running SPs in the VPC network. Using remote attestation, the tenant provisions the sensitive information securely to SGX-VPN securely. The participating entities in SGX-VPN exchanges the SAs for packet processing securely using local and remote attestation. We evaluate the algorithms in SGX-VPN using Scyther, a formal analysis tool, to prove the security of SGX-VPN. We implement a prototype on the actual SGX-enabled machine to measure the performance overhead of SGX-VPN in the packet processing.

We also 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.

### 6.2 Future Work

The literature of cloud VPN and enclave migration are relatively new, and there are still challenging problems to be addressed to accelerate the deployment of SGX-VPN and eMotion into cloud computing.

The performance of the SGX-VPN prototype decreases drastically against the smaller packets. Because the number of packets grows as the packets shorten for the given input traffic, the number of enclave transitions also increases. This increment yields to the decrement of the performance. We will apply Switchless Calls [115] that eliminates enclave transitions to SPP for guaranteeing the enhanced performance for short packets.

The number of remote verifications on SPs in SGX-VPN can increase depending on the number of SPP in the VPC network. Because each verification involves interworking with IAS for remote attestation, it is necessary to centralize the remote verifications to a single component in the VPC network with regard to the tenant's management efforts. We will study about a mechanism to adopt the 3rd party attestation service [116], which enables non-Intel parties to build their own attestation infrastructure, to SGX-VPN. Using this extension, it is possible to implement a single point in the VPC network that

manages remote verifications on SPs.

eMotion mainly focuses on the extensions of SGX implementation to enable managed enclave migration. The prototype based on OpenSGX confirm 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. We will research other SGX emulators like S-OpenSGX [117] that runs 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.

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. We will study about migrating the sealed data of the enclave securely.

## Bibliography

- [1] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, pages 13:1–13:7, 2013.
- [2] Jeongseok Son, Yongqiang Xiong, Kun Tan, Paul Wang, Ze Gan, and Sue Moon. Protego: Cloud-Scale Multitenant IPsec Gateway. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 473–485. USENIX Association, 2017.
- [3] Steffen Schulz and Ahmad-Reza Sadeghi. Secure VPNs for Trusted Computing Environments. In *International Conference on Trusted Computing*, pages 197–216. Springer, 2009.
- [4] AWS Managed VPN Connections. [https://docs.aws.amazon.com/vpc/latest/userguide/VPC\\_VPN.html](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_VPN.html), last accessed 2018-10-02.
- [5] Azure VPN Gateway. <https://azure.microsoft.com/en-us/services/vpn-gateway/>, last accessed 2018-10-02.
- [6] Google Cloud VPN. <https://cloud.google.com/vpn/docs/concepts/overview>, last accessed 2018-10-02.
- [7] Carlos V Rozas, Mona Vij, Rebekah M Leslie-Hurd, Krystof C Zmudzinski, Somnath Chakrabarti, Francis X McKeen, Vincent R Scarlata, Simon P Johnson, Ilya Alexandrovich, Gilbert Neiger, et al. Processors, methods, systems, and instructions to support live migration of protected containers, 2017. US Patent 9,710,401.
- [8] Carlos V Rozas, Mona Vij, Rebekah M Leslie-Hurd, Krystof C Zmudzinski, Somnath Chakrabarti, Francis X McKeen, Vincent R Scarlata, Simon P Johnson, and Ilya Alexandrovich. Platform migration of secure enclaves, 2018. US Patent 9,942,035.
- [9] Prerit Jain, Soham Desai, Seongmin Kim, Ming-Wei Shih, JaeHyuk Lee, Changho Choi, Youjung Shin, Taesoo Kim, Brent Byunghoon Kang, and Dongsu Han. OpenSGX: An Open Platform for SGX Research. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.
- [10] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.
- [11] Michael R. Hines and Kartik Gopalan. 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*, pages 51–60. ACM, 2009.
- [12] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.



- [13] Stephen Kent and Karen Seo. Security Architecture for the Internet Protocol. <https://tools.ietf.org/html/rfc4301>, 2005. last accessed 2018-12-05.
- [14] Mina Tahmasbi Arashloo, Pavel Shirshov, Rohan Gandhi, Guohan Lu, Lihua Yuan, and Jennifer Rexford. A Scalable VPN Gateway for Multi-Tenant Cloud Services. *ACM SIGCOMM Computer Communication Review*, 48(1):49–55, 2018.
- [15] CVE-2015-3340: Information leak through XEN.DOMCTL\_gettscinfo. <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3340>, last accessed 2018-04-29.
- [16] C Kaufman, P Hoffman, Y Nir, and P Eronen. Internet Key Exchange (IKEv2) Protocol. <https://tools.ietf.org/html/rfc5996>, 2010. last accessed 2018-12-26.
- [17] Cas Cremers. Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In *European Symposium on Research in Computer Security*, pages 315–334. Springer, 2011.
- [18] Natalia C Fernandes, Marcelo DD Moreira, Igor M Moraes, Lino Henrique G Ferraz, Rodrigo S Couto, Hugo ET Carvalho, Miguel Elias M Campista, Luís Henrique MK Costa, and Otto Carlos MB Duarte. Virtual networks: isolation, performance, and trends. *annals of telecommunications-Annales des télécommunications*, 66(5-6):339–355, 2011.
- [19] Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. Panoply: Low-TCB Linux Applications With SGX Enclaves. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017.
- [20] Cas J.F. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *International Conference on Computer Aided Verification*, pages 414–418. Springer, 2008.
- [21] Data Plane Development Kit. <https://www.dpdk.org/>, last accessed 2018-10-04.
- [22] Intel Corporation. SGX Virtualization. <https://01.org/intel-software-guard-extensions/sgx-virtualization>, 2016. last accessed 2018-04-29.
- [23] Intel Corporation. Intel Software Guard Extensions Developer Guide. [https://01.org/sites/default/files/documentation/intel\\_sgx\\_developer\\_guide\\_pdf.pdf](https://01.org/sites/default/files/documentation/intel_sgx_developer_guide_pdf.pdf), 2016. last accessed 2018-04-29.
- [24] Jinyu Gu, Zhichao Hua, Yubin Xia, Haibo Chen, Binyu Zang, Haibing Guan, and Jinming Li. Secure Live Migration of SGX Enclaves on Untrusted Cloud. In *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*, pages 225–236. IEEE, 2017.
- [25] Fritz Alder, Arseny Kurnikov, Andrew Paverd, and N Asokan. Migrating SGX Enclaves with Persistent State. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 195–206. IEEE, 2018.
- [26] Stephen Kent. IP Encapsulating Security Payload (ESP). <https://tools.ietf.org/html/rfc4303>, 2005. last accessed 2018-11-17.
- [27] Stephen Kent. IP Authentication Header (AH). <https://tools.ietf.org/html/rfc4302>, 2005. last accessed 2018-12-06.

- [28] Pasi Eronen and Paul Hoffman. IKEv2 Clarifications and Implementation Guidelines. <https://tools.ietf.org/html/rfc4718>, 2006. last accessed 2018-10-27.
- [29] Intel Corporation. Intel Software Guard Extensions Programming Reference. <https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf>, 2014. last accessed 2018-04-29.
- [30] Victor Costan and Srinivas Devadas. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086, 2016. <https://eprint.iacr.org/2016/086>, last accessed 2018-05-04.
- [31] Shay Gueron. A Memory Encryption Engine Suitable for General Purpose Processors. Cryptology ePrint Archive, Report 2016/204, 2016. last accessed 2018-12-09.
- [32] Data Plane Development Kit Programmer’s Guide. [http://doc.dpdk.org/guides/prog\\_guide/](http://doc.dpdk.org/guides/prog_guide/), last accessed 2018-12-09.
- [33] Assad Abbas and Samee U Khan. A Review on the State-of-the-Art Privacy-Preserving Approaches in the e-Health Clouds. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1431–1441, 2014.
- [34] P Getzi Jeba Leelipushpam and J Sharmila. Live VM migration techniques in cloud environment—a survey. In *2013 IEEE Conference on Information & Communication Technologies*, pages 408–413. IEEE, 2013.
- [35] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 457–473. Springer, 2005.
- [36] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, pages 89–98. Acm, 2006.
- [37] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *Proceeding of 2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE, 2007.
- [38] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical Techniques for Searches on Encrypted Data. In *Proceeding of 2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE, 2000.
- [39] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption with Keyword Search. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 506–522. Springer, 2004.
- [40] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. *Journal of Cryptology*, 26(2):191–224, 2013.
- [41] Elaine Shi and Brent Waters. Delegating Capabilities in Predicate Encryption Systems. In *International Colloquium on Automata, Languages, and Programming*, pages 560–578. Springer, 2008.
- [42] Dan Boneh and Matt Franklin. Identity-Based Encryption from the Weil Pairing. In *Annual International Cryptology Conference*, pages 213–229. Springer, 2001.

- [43] Craig Gentry and Alice Silverberg. Hierarchical ID-Based Cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 548–566. Springer, 2002.
- [44] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 127–144. Springer, 1998.
- [45] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178. ACM, 2009.
- [46] Mehdi Bahrani and Mukesh Singhal. A Light-Weight Permutation Based Method for Data Privacy in Mobile Cloud Computing. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 189–198. IEEE, 2015.
- [47] Xun Yi, Fang-Yu Rao, Elisa Bertino, and Athman Bouguettaya. Privacy-Preserving Association Rule Mining in Cloud Computing. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 439–450. ACM, 2015.
- [48] Jun Zhou, Zhenfu Cao, Xiaolei Dong, and Xiaodong Lin. PPDM: A Privacy-Preserving Protocol for Cloud-Assisted e-Healthcare Systems. *IEEE Journal of Selected Topics in Signal Processing*, 9(7):1332–1344, 2015.
- [49] Syam Kumar Pasupuleti, Subramanian Ramalingam, and Rajkumar Buyya. An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing. *Journal of Network and Computer Applications*, 64:12–22, 2016.
- [50] Hong Rong, Hui-Mei Wang, Jian Liu, and Ming Xian. Privacy-Preserving k-Nearest Neighbor Computation in Multiple Cloud Environments. *IEEE Access*, 4:9589–9603, 2016.
- [51] Issa Khalil, Abdallah Khreishah, and Muhammad Azeem. Consolidated Identity Management System for secure mobile cloud computing. *Computer Networks*, 65:99–110, 2014.
- [52] Wei Wei, Fengyuan Xu, and Qun Li. Mobishare: Flexible privacy-preserving location sharing in mobile online social networks. In *2012 IEEE Conference on Computer Communications (INFOCOM)*, pages 2616–2620. IEEE, 2012.
- [53] Ben Niu, Qinghua Li, Xiaoyan Zhu, Guohong Cao, and Hui Li. Enhancing privacy through caching in location-based services. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1017–1025. IEEE, 2015.
- [54] Fang-Yie Leu, Jia-Chun Lin, Ming-Chang Li, Chao-Tung Yang, and Po-Chi Shih. Integrating grid with intrusion detection. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, volume 1, pages 304–309. IEEE, 2005.
- [55] Chi-Chun Lo, Chun-Chieh Huang, and Joy Ku. A cooperative intrusion detection system framework for cloud computing networks. In *2010 39th International Conference on Parallel Processing Workshops*, pages 280–284. IEEE, 2010.

- [56] Claudio Mazzariello, Roberto Bifulco, and Roberto Canonico. Integrating a network IDS into an open source Cloud Computing environment. In *2010 Sixth International Conference on Information Assurance and Security*, pages 265–270. IEEE, 2010.
- [57] Eucalyptus. <https://www.eucalyptus.cloud/>. last accessed 2019-04-20.
- [58] Martin Roesch. Snort: Lightweight intrusion detection for networks. volume 99, pages 229–238, 1999.
- [59] Ahmad-Reza Sadeghi and Steffen Schulz. Extending IPsec for Efficient Remote Attestation. In *International Conference on Financial Cryptography and Data Security*, pages 150–165. Springer, 2010.
- [60] Ingo Bente, Bastian Hellmann, Joerg Vieweg, Josef von Helden, and Arne Welzel. Interoperable Remote Attestation for VPN Environments. In *International Conference on Trusted Systems*, pages 302–315. Springer, 2010.
- [61] TCG. TPM Main Specification. <https://trustedcomputinggroup.org/resource/tpm-main-specification/>, 2019. last accessed 2019-02-01.
- [62] TCG Trusted Network Communications Work Group. TNC Architecture for Interoperability. <https://trustedcomputinggroup.org/wp-content/uploads/TCG-TNC-Architecture-for-Interoperability-Version-2.0-Revision-13-.pdf>, 2017. last accessed 2019-02-06.
- [63] Mazhar Ali, Samee U Khan, and Athanasios V Vasilakos. Security in cloud computing: Opportunities and challenges. *Information sciences*, 305:357–383, 2015.
- [64] Boris Danev, Ramya Jayaram Masti, Ghassan O Karame, and Srdjan Capkun. Enabling secure VM-vTPM migration in private clouds. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC)*, pages 187–196. ACM, 2011.
- [65] Xin Wan, Xinfang Zhang, Liang Chen, and Jianxin Zhu. An improved vTPM migration protocol based trusted channel. In *2012 International Conference on Systems and Informatics (ICSAI2012)*, pages 870–875. IEEE, 2012.
- [66] Mudassar Aslam, Christian Gehrman, and Mats Björkman. Security and Trust Preserving VM Migrations in Public Clouds. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 869–876. IEEE, 2012.
- [67] MR Anala, Jyoti Shetty, and G Shobha. A framework for secure live migration of virtual machines. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 243–248. IEEE, 2013.
- [68] Jaemin Park, Sungjin Park, Jisoo Oh, and Jongjin Won. Toward Live Migration of SGX-Enabled Virtual Machines. In *2016 IEEE World Congress on Services (SERVICES)*, pages 111–112. IEEE, 2016.
- [69] Jim Gordon. Microsoft Azure Confidential Computing with Intel SGX. <https://software.intel.com/en-us/blogs/2018/11/08/microsoft-azure-confidential-computing-with-intel-sgx>, 2018. last accessed 2019-04-22.



- [70] Pratheek Karnati. Data-in-use protection on IBM Cloud using Intel SGX. <https://www.ibm.com/blogs/bluemix/2018/05/data-use-protection-ibm-cloud-using-intel-sgx/>, 2018. last accessed 2019-04-22.
- [71] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Transactions on Computer Systems (TOCS)*, 33(3):8:1–8:26, 2015.
- [72] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O’Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eysers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symp. Operating Systems Design and Implementation*, volume 16, pages 689–703, 2016.
- [73] Chia-Che Tsai, Donald E Porter, and Mona Vij. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC)*, 2017.
- [74] Hongliang Tian, Yong Zhang, Chunxiao Xing, and Shoumeng Yan. SGXKernel: A Library Operating System Optimized for Intel SGX. In *Proceedings of the Computing Frontiers Conference*, pages 35–44. ACM, 2017.
- [75] Donald E Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C Hunt. Rethinking the library OS from the top down. In *ACM SIGPLAN Notices*, volume 46, pages 291–304. ACM, 2011.
- [76] Chia-Che Tsai, Kumar Saurabh Arora, Nehal Bandi, Bhushan Jain, William Jannen, Jitin John, Harry A Kalodner, Vrushali Kulkarni, Daniela Oliveira, and Donald E Porter. Cooperation and security isolation of library OSes for multi-process applications. In *Proceedings of the Ninth European Conference on Computer Systems*, page 9. ACM, 2014.
- [77] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. SecureKeeper: Confidential ZooKeeper using Intel SGX. In *Proceedings of the 17th International Middleware Conference*, page 14. ACM, 2016.
- [78] Stefan Brenner, Colin Wulf, and Rüdiger Kapitza. Running ZooKeeper Coordination Services in Untrusted Clouds. In *10th Workshop on Hot Topics in System Dependability (HotDep 14)*, 2014.
- [79] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *2015 IEEE Symposium on Security and Privacy*, pages 38–54. IEEE, 2015.
- [80] Apache ZooKeeper. <https://zookeeper.apache.org/>. last accessed 2019-04-23.
- [81] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [82] Apache Hadoop. <https://hadoop.apache.org/>. last accessed 2019-04-23.
- [83] Seongmin Kim, Youjung Shin, Jaehyung Ha, Taesoo Kim, and Dongsu Han. A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, pages 7:1–7:7. ACM, 2015.



- [84] Ming-Wei Shih, Mohan Kumar, Taesoo Kim, and Ada Gavrilovska. S-NFV: Securing NFV states by using SGX. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 45–48. ACM, 2016.
- [85] Michael Coughlin, Eric Keller, and Eric Wustrow. Trusted Click: Overcoming Security Issues of NFV in the Cloud. In *Proceedings of the 2017 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 31–36. ACM, 2017.
- [86] Juhyeong Han, Seongmin Kim, Jaehyeong Ha, and Dongsu Han. SGX-Box: Enabling Visibility on Encrypted Traffic Using a Secure Middlebox Module. In *Proceedings of the First Asia-Pacific Workshop on Networking*, pages 99–105. ACM, 2017.
- [87] Huayi Duan, Xingliang Yuan, and Cong Wang. LightBox: Full-stack Protected Stateful Middlebox at Lightning Speed. *arXiv preprint arXiv:1706.06261*, 2018.
- [88] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. SafeBricks: Shielding Network Functions in the Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI’18)*, 2018.
- [89] David Goltzsche, Signe Rüsch, Manuel Nieke, Sébastien Vaucher, Nico Weichbrodt, Valerio Schiavoni, Pierre-Louis Aublin, Paolo Cosa, Christof Fetzer, Pascal Felber, et al. EndBox: Scalable Middlebox Functions Using Client-Side Trusted Execution. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 386–397. IEEE, 2018.
- [90] Ketan Bhardwaj, Ming-Wei Shih, Pragya Agarwal, Ada Gavrilovska, Taesoo Kim, and Karsten Schwan. Fast, Scalable and Secure Onloading of Edge Functions Using AirBox. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 14–27. IEEE, 2016.
- [91] Seong Min Kim, Juhyeong Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. Enhancing Security and Privacy of Tor’s Ecosystem by Using Trusted Execution Environments. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI’17)*, pages 145–161, 2017.
- [92] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.
- [93] Passive Real-time Asset Detection System. <https://github.com/gamlinux/prads>. last accessed 2018-11-28.
- [94] libntoh. <https://github.com/sch3m4/libntoh>. last accessed 2018-11-28.
- [95] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. NetBricks: Taking the V out of NFV. In *OSDI*, pages 203–216, 2016.
- [96] Intel Corporation. Attestation Service for Intel Software Guard Extensions (Intel SGX): API Documentation. <https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf>. last accessed 2018-05-22.
- [97] Intel SGX SSL. <https://github.com/intel/intel-sgx-ssl/>. last accessed 2019-01-23.
- [98] OpenSSL Project. <https://www.openssl.org/>. last accessed 2019-01-23.

- [99] Cas Cremers and Sjouke Mauw. Operational Semantics of Security Protocols. In *Scenarios: Models, Transformations and Tools*, pages 66–89. Springer, 2005.
- [100] Ofir Weisse, Valeria Bertacco, and Todd Austin. Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 81–93. ACM, 2017.
- [101] Intel Software Guard Extensions for Linux OS. <https://github.com/intel/linux-sgx>. last accessed 2019-01-23.
- [102] Jean Paul Degabriele and Kenneth G Paterson. Attacking the IPsec Standards in Encryption-only Configurations. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 335–349. IEEE, 2007.
- [103] Jean Paul Degabriele and Kenneth G Paterson. On the (In)Security of IPsec in MAC-then-Encrypt Configurations. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, pages 493–504. ACM, 2010.
- [104] Nico Weichbrodt, Pierre-Louis Aublin, and Rüdiger Kapitza. sgx-perf: A Performance Analysis Tool for Intel SGX Enclaves. In *Proceedings of the 19th International Middleware Conference*, pages 201–213. ACM, 2018.
- [105] Scott Bradner and Jim McQuaid. Benchmarking Methodology for Network Interconnect Devices. <https://tools.ietf.org/html/rfc2544>, 1999. last accessed 2019-03-26.
- [106] Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. Rollback and Forking Detection for Trusted Execution Environments using Lightweight Collective Memory. In *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*, pages 157–168. IEEE, 2017.
- [107] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transaction on Information Theory*, 22(6):644–654, 1976.
- [108] Lily Chen. Recommendation for Key Derivation Using Pseudorandom Functions. <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-108.pdf>, 2009. last accessed 2018-05-12.
- [109] Morris J Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. [https://www.nist.gov/publication/get\\_pdf.cfm?pub\\_id=51288](https://www.nist.gov/publication/get_pdf.cfm?pub_id=51288), 2007. last accessed 2018-04-29.
- [110] Somnath Chakrabarti, Rebekah Leslie-Hurd, Mona Vij, Frank McKeen, Carlos Rozas, Dror Caspi, Ilya Alexandrovich, and Ittai Anati. Intel Software Guard Extensions (Intel SGX) Architecture for Oversubscription of Secure Memory in a Virtualized Environment. In *Proceedings of the Hardware and Architectural Support for Security and Privacy*, pages 7:1–7:8. ACM, 2017.
- [111] PolarSSL Project. <https://www.polarssl.org/>. last accessed 2018-04-29.
- [112] Shashank Sahni and Vasudeva Varma. A Hybrid Approach To Live Migration Of Virtual Machines. In *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 1–5. IEEE, 2012.

- [113] James Greene. Intel Trusted Execution Technology - Hardware-based Technology for Enhancing Server Platform Security. <https://www.intel.de/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf>, 2012. last accessed 2018-04-29.
- [114] TCG. TCG PC Client Specific Implementation Specification for Conventional BIOS. [https://www.trustedcomputinggroup.org/wp-content/uploads/TCG\\_PCClientImplementation\\_1-21\\_1\\_00.pdf](https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_PCClientImplementation_1-21_1_00.pdf), 2012. last accessed 2018-04-29.
- [115] Hongliang Tian, Qiong Zhang, Shoumeng Yan, Alex Rudnitsky, Liron Shacham, Ron Yariv, and Noam Milshten. Switchless Calls Made Practical in Intel SGX. In *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, pages 22–27. ACM, 2018.
- [116] Intel Corporation. Supporting Third Party Attestation for Intel SGX with Intel Data Center Attestation Primitives. <https://software.intel.com/sites/default/files/managed/f1/b8/intel-sgx-support-for-third-party-attestation.pdf>, 2018. last accessed 2019-03-23.
- [117] Changho Choi, Nohyun Kwak, Jinsoo Jang, Daehee Jang, Kuenwhee Oh, Kyungsoo Kwag, and Brent Byunghoon Kang. S-OpenSGX: a system-level platform for exploring SGX enclave-based computing. *Computer & Security*, 70:290–306, 2017.