DEGREE PROJECT IN ELECTRICAL ENGINEERING, SECOND CYCLE
STOCKHOLM, SWEDEN 2017

Trusted Execution Environments for Open vSwitch

A security enabler for the 5G mobile network

KHALID ELBASHIR

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Trusted Execution Environments for Open vSwitch

A security enabler for the 5G mobile network

Khalid Elbashir

2017-11-17

Master's Thesis

Examiner
Gerald Q. Maguire Jr.

Academic adviser
Anders Västberg

Industrial adviser
Nicolae Paladi, RISE SICS

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

The advent of virtualization introduced the need for virtual switches to interconnect virtual machines deployed in a cloud infrastructure. With Software Defined Networking (SDN), a central controller can configure these virtual switches. Virtual switches execute on commodity operating systems. Open vSwitch is an open source project that is widely used in production cloud environments. If an adversary gains access with full privileges to the operating system hosting the virtual switch, then Open vSwitch becomes vulnerable to a variety of different attacks that could compromise the whole network.

The purpose of this thesis project is to improve the security of Open vSwitch implementations in order to ensure that only authenticated switches and controllers can communicate with each other, while maintaining code integrity and confidentiality of keys and certificates. The thesis project proposes a design and shows an implementation that leverages Intel® Safe Guard Extensions (SGX) technology. A new library, TLSonSGX, is implemented. This library replaces the use of the OpenSSL library in Open vSwitch. In addition to implementing standard Transport Level Security (TLS) connectivity, TLSonSGX confines TLS communication in the protected memory enclave and hence protects TLS sensitive components necessary to provide confidentiality and integrity, such as private keys and negotiated symmetric keys. Moreover, TLSonSGX introduces new, secure, and automatic means to generate keys and obtain signed certificates from a central Certificate Authority that validates using Linux Integrity Measurements Architecture (IMA) that the Open vSwitch binaries have not been tampered with before issuing a signed certificate. The generated keys and obtained certificates are stored in the memory enclave and hence never exposed as plaintext outside the enclave. This new mechanism is a replacement for the existing manual and unsecure procedures (as described in Open vSwitch project).

A security analysis of the system is provided as well as an examination of performance impact of the use of a trusted execution environment. Results show that generating keys and certificates using TLSonSGX takes less than 0.5 seconds while adding 30% latency overhead for the first packet in a flow compared to using OpenSSL when both are executed on Intel® Core™ i7-6600U processor clocked at 2.6 GHz. These results show that TLSonSGX can enhance Open vSwitch security and reduce its TLS configuration overhead.

Keywords Open vSwitch, SDN, Trusted Execution Environment, SGX, Cloud Computing

Sammanfattning

Framkomsten av virtualisering införde behovet av virtuella växlar för att koppla tillsammans virtuella maskiner placerade i molninfrastruktur. Med mjukvarubaserad nätverksteknik (SDN), kan ett centralt styrenhet konfigurera dessa virtuella växlar. Virtuella växlar kör på standardoperativsystem. Open vSwitch är ett open-source projekt som ofta används i molntjänster. Om en motståndare får tillgång med fullständiga privilegier till operativsystemet där Open vSwitch körs, blir Open vSwitch utsatt för olika attacker som kan komprometttera hela nätverket. Syftet med detta examensarbete är att förbättra säkerheten hos Open vSwitch för att garantera att endast autentiserade växlar och styrenheter kan kommunicera med varandra, samtidigt som att upprätthålla kod integritet och konfidentialitet av nycklar och certifikat.

Detta examensarbete föreslår en design och visar en implementation som andvändar Intel®s Safe Guard Extensions (SGX) teknologi. Ett nytt bibliotek, TLSonSGX, är implementerat. Detta bibliotek ersätter biblioteket OpenSSL i Open vSwitch. Utöver att det implementerar ett standard “Transport Layer Security” (TLS) anslutning, TLSonSGX begränsar TLS kommunikation i den skyddade minnes enklaven och skyddar därför TLS känsliga komponenter som är nödvändiga för att ge sekretess och integritet, såsom privata nycklar och förhandlade symmetriska nycklar. Dessutom introducerar TLSonSGX nya, säkra och automatiska medel för att generera nycklar och få signerade certifikat från en central certifikatmyndighet som validerar, med hjälp av Linux Integrity Measurements Architecture (IMA), att Open vSwitch-binärerna inte har manipulerats innan de utfärdade ett signerat certifikat. De genererade nycklarna och erhållna certifikat lagras i minnes enklaven och är därför aldrig utsatta utanför enklaven. Denna nya mekanism ersätter de manuella och osäkra procedurerna som beskrivs i Open vSwitch projektet.

En säkerhetsanalys av systemet ges såväl som en granskning av prestandaffekten av användningen av en pålitlig exekveringsmiljö. Resultaten visar att använda TLSonSGX för att generera nycklar och certifikat tar mindre än 0,5 sekunder medan det lägger 30% latens overhead för det första paketet i ett flöde jämfört med att använda OpenSSL när båda exekveras på Intel ® Core ™ processor i7-6600U klockad vid 2,6 GHz. Dessa resultat visar att TLSonSGX kan förbättra Open vSwitch säkerhet och minska TLS konfigurationskostnaden.

Nyckelord Open vSwitch, SDN, mjukvarudefinierade nätverk, SGX, molntjänster

Acknowledgements

First of all, I would like to express my gratitude to my examiner Professor Gerald Q. Maguire Jr. for his invaluable feedback and guidance throughout the course of this thesis project.

I would like to thank my supervisor from RI.SE SICS Security Lab, Dr. Nicolae Paladi for giving me the opportunity to work on this thesis project and his continuous support.

Special thanks to my wife, Marwa, for her support and understanding, my parents, and to all my friends and in particular Mina Tawfik for the useful discussions and suggestions.

Stockholm, November 2017
Khalid Elbashir

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
ARM	Advanced RISC Machine
BIOS	Basic Input/Output System
CA	Certificate Authority
CBC	Cipher Block Chaining
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
ECALL	Call from Application to the Enclave
ECDHE	Elliptic Curve Ephemeral Diffie Hellman
EPC	Enclave Page Cache
EPID	Enhanced Privacy ID
GRE	Generic Routing Encapsulation
IAS	Intel® Attestation Service
IMA	Linux Integrity Measurement Architecture
IP	Internet Protocol
IQR	Interquartile Range
KVM	Kernel-based Virtual Machine
L2	Layer 2
L3	Layer 3
L4	Layer 4
LSM	Linux Security Modules

MAC	Message Authentication Code
MTU	Maximum Transmission Unit
NIC	Network Interface Controller
OCALL	Call from Enclave to the Application
OVS	Open vSwitch
OVSDB	Open vSwitch Database Management Protocol
PCR	Platform Configuration Register
pps	Packets Per Second
PTP	Precision Time Protocol
RAM	Random Access Memory
REE	Rich Execution Environment
RSA	Rivest–Shamir–Adleman
SDK	Software Development Kit
SDN	Software Defined Networking
SGX	Safe Guard Extensions
SHA-1	Secure Hash Algorithm 1
SMACK	Simplified Mandatory Access Control Kernel
SSL	Secure Sockets Layer
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
TXT	Trusted Execution Technology
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VMX	Virtual Machine Extensions
vNIC	Virtual NIC
VXLAN	Virtual Extensible Local Area Network

1 Introduction

This chapter gives a brief background, description of the problem, and the goals of this thesis project. The chapter concludes with the structure of the thesis.

1.1 Background

Software Defined Networking (SDN) is a network architecture that separates the data and control planes. Through the use of a central controller, it is possible for network operators to have a global view of the network and configure any of the SDN switches in it.

The advent of virtualization introduced the need for virtual switches to interconnect the virtual machines deployed in the cloud. Nowadays, virtual switches are well integrated with cloud infrastructures supporting scaling of virtual machines, isolation between cloud tenants, and rich packet forwarding capabilities.

1.2 Problem Definition

In a cloud infrastructure, virtual switches are fundamental elements as they interconnect virtual machines. Both the virtual switches and virtual machines execute on commodity operating systems which make them vulnerable to different types of attacks that could compromise the whole network. Avoiding and preventing these attacks is the core of the problem this thesis project addresses. More specifically, this thesis investigates how to strengthen the security of virtual switches in an SDN based cloud infrastructure.

1.3 Purpose

The purpose of this thesis is to improve the security of an Open vSwitch implementation by identifying security sensitive components and confining them to a trusted execution environment as well as enabling both the SDN controller and verified virtual switches to establish and maintain authenticated secure communication channels.

This improvement increases the security of the control plane, thus indirectly increasing trust in the dataplane's reliability and as a result helping cloud infrastructure providers meet their service level agreements.

1.4 Goals

The goal of this thesis project is to strengthen the security and isolation of networks in cloud infrastructures. This goal is divided into the following four sub-goals:

1. Perform a background study of a Trusted Execution Environment (TEE) suitable for this thesis project.
2. Perform an analysis of Open vSwitch's architecture in order to identify security sensitive components that can be confined to a TEE.
3. Find out how to establish secure connections between SDN controllers and virtual switches, such that connections can only be established from *verified* virtual switches[†] towards authenticated SDN controllers.
4. Design and implement a prototype and evaluate it from both security and performance perspectives.

1.5 Research Methodology

The thesis project utilizes a mix of applied and experimental research methods to achieve the goals stated in Section 1.4. More details about the choice of research methodology are given in Chapter 4.

1.6 Delimitations

The focus of this thesis project is virtual switches deployed on top of x86 architecture commodity hardware; therefore, physical switches and other architectures are out of the scope of this project. Although there are many virtual switch implementations, this thesis project will consider only one open source project: Open vSwitch.

The system model is limited to have only one SDN controller in the network. The adversary's capabilities are limited to those described in Section 3.1. Finally, the implementation focusses on addressing the functional requirements described in Section 5.1.

[†]This is explained further in Section 5.1

1.7 Structure of the thesis

Chapter 2 introduces the necessary technical and theoretical background underlying this thesis along with related work. Chapter 3 discusses the adversary model and threat analysis. Chapter 4 discusses the method used to carry out this thesis, while Chapter 5 presents in detail the functional requirements, the design of the framework, and the implementation of the prototype.

Chapter 6 presents the results of the evaluation of the framework with regard to its adherence to the requirements, performance measurements, and security analysis of the framework. Finally, Chapter 7 states the limitations of this thesis project, suggests future work, offers some reflections, and states some conclusions.

2 Background

This chapter introduces the necessary theoretical and technical background for this thesis. Section 2.1 gives background information about SDN. Then Section 2.2 introduces virtual switches with a specific focus on Open vSwitch and discusses its different components. Section 2.3 describes different TEEs and then focuses on Intel® Safe Guard Extensions (SGX). Linux Integrity Measurement Architecture (IMA) is described in Section 2.4. Finally, Section 2.5 discusses related work.

2.1 Software Defined Networking

SDN is a networking architecture where control and data planes are separated. This makes it possible to have a centralized controller where forwarding and routing decisions are made and then these decisions are conveyed to the elements that realize the data plane. Having a centralized controller enables easy configuration of the whole network, despite the fact that this network could be composed of equipment from multiple vendors. SDN provides an easy way to continuously improve, update, and deploy new networking software and services on top of the controller [1].

OpenFlow [2] has emerged as a widely used protocol between SDN controllers and dataplane switches. A switch that supports OpenFlow has a flow table. OpenFlow provides a standardized protocol to add or remove flows from this table. Each rule in the flow table consists of a flow and an associated action. Flows are identified by matching all or particular fields of Layer 2 (L2), Layer 3 (L3), and Layer 4 (L4) headers along with the input port of the switch. Moreover, flows can be either an exact match or wildcard match.

OpenFlow supports different actions, such as forward to a port, send packet to controller, drop packets, or set certain values in the headers [3]. The flow table also maintains priority and statistics (number of packets and bytes) per flow in the table as well as the time the flow was last active (i.e. the time since the last packet in this flow).

The adoption of OpenFlow has led to the introduction of many controller platforms. Some of the most widely used open source controllers are POX, Ryu, FloodLight, and OpenDayLight as per the survey in [4].

2.2 Virtual Switches

Virtualization of computers introduced the need for a dataplane layer to interconnect the different virtual machines. Virtual switches are implemented in software and deployed in servers providing both inter and intra server connectivity for virtual machines. Moreover, virtual switches enable network virtualization, by providing a network abstraction decoupled from the underlying physical network that can meet requirements, such as scaling the network to interconnect hundreds of virtual machines and to provide isolation between different tenants [5]. Virtual switches can also interoperate with hypervisors and provide rich forwarding and routing capabilities based on information inferred from virtual machines' properties .

In a cloud infrastructure, as depicted in Figure 2.1 and discussed in [6], the cloud orchestrater invokes the SDN controller to connect virtual machines to virtual switches. The SDN controller then configures the virtual switches (This occurs via the communication shown as dashed lines. It is this communication that is the focus of this thesis project). This configuration includes port settings, L2/L3 addresses of corresponding virtual machines, and the flows necessary to connect virtual machines and external networks. Virtual machines that belong to different tenants are isolated by placing them in different logical networks.

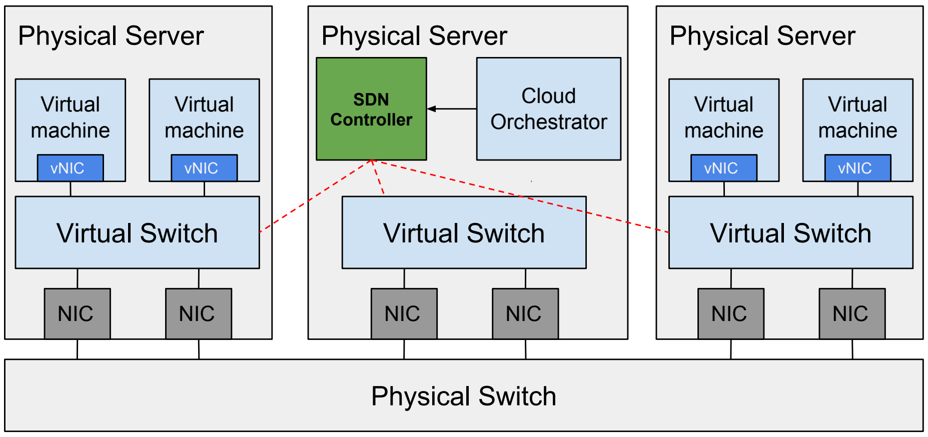


Figure 2.1: SDN in a cloud infrastructure

There are many virtual switches implementations. Linux bridge [7] is an Ethernet switch implemented in the kernel that supports traffic filtering and shaping. KVM [8] utilizes Linux bridge to provide connectivity between virtual machines residing in the same server as well as connectivity to external networks. Open vSwitch (OVS) [5] operates as both an Ethernet switch and an OpenFlow switch. We will discuss OVS in Section 2.2.1.

2.2.1 Open vSwitch

Open vSwitch (OVS) is an open source software switch that is widely used in production environments and it is supported by many different Linux distributions. In addition, it has been ported to Microsoft's Windows operating systems [9].

Typically, OVS is deployed in the host and works with hypervisors (for example, KVM) and container systems (such as Docker) and it interconnects the virtual machines/containers and can utilize the physical network interfaces of the server host.

In addition to operating as a standard Ethernet switch, OVS acts as an OpenFlow switch by exporting an external interface that uses the OpenFlow protocol. Through OpenFlow, OVS can send packets to the SDN controller and receive flow table updates at runtime from the controller [5]. OVS exports another external interface using the OVSDB management protocol. This enables reading a switch's configuration, creating or deleting switches, and changing switch configurations (for example adding or deleting ports and configuring the quality of service). Both OpenFlow and OVSDB support secure communication using the Transport Layer Security (TLS) protocol.

Open vSwitch supports several different protocols (such as Virtual Local Area Network (VLAN), Generic Routing Encapsulation (GRE), and Virtual Extensible Local Area Network (VXLAN)) to isolate different tenants within cloud environments.

An Open vSwitch implementation consists of three components [10] as shown in Figure 2.2:

- **Slow Path (ovs-vswitchd)**
  The slow path is implemented in user space by the `ovs-vswitchd` process. It comprises the forwarding logic and has an OpenFlow interface. It also maintains the flow table. The flow table can be manipulated manually by using the `ovs-ofctl` command or by the external SDN controller using OpenFlow. Using OpenFlow, the SDN controller can monitor flows, get statistics for flows, and send packets to the switch.

- **Fast Path (Data Path)**
  The fast path, also known as the data path, was initially implemented as a kernel module. The fast path is where packet forwarding happens. The data path maintains a cache of the flow table, while `ovs-vswitchd` uses Netlink sockets to update this cache table with flows and associated actions.

  Packets arriving from a physical or virtual Network Interface Cards (NICs)[†] are received by the data path. The data path searches its cached flow table for a matching flow, otherwise it forwards the packet over the Netlink socket to `ovs-vswitchd` for it to decide how the packet will be handled. Upon receiving the packet the `ovs-vswitchd` process will send the needed update for the cache table along with the original packet that will now be forwarded based upon the new entry in the cached flow table. As a result, subsequent packets in the same flow will be handled completely by the data path. When ovs-vswitch does not find a match in its flow table, it sends the packet to the SDN controller which decides how the packet is to be processed and then sends an update to the flow table along with the original packet and an action to forward it according to the newly installed flow table entry [11]. The packet flow in fast path and slow path are shown in Figure 2.3. The cached flow table can also be accessed using the `ovs-dpctl` command.

  Initially, the cached flow table was designed to support microflow caching, i.e. exact matches on all packet header fields. However, this caused performance degradation when deployed with a large number of short lived connections [10]. As a result, megaflow caching was introduced in which flows can be aggregated leading to a two-level cache.

  The introduction of the Data Plane Development Kit (DPDK)[‡] enabled the data path to be implemented in user space, thus providing high performance packet processing. This results in higher network throughput and lower latency [12]. The inter process communication between the two user space processes (`ovs-vswitchd` and data path) is done using a ring data structure in shared memory.

- **Configuration Database (ovsdb-server)**
  The ovsdb-server maintains a persistent database where all switch

[†]The term NIC is used to refer to a network interface even where there is not actually a circuit card involved.

[‡]DPDK was initially introduced by Intel®, but it is now a Linux Foundation Project.

configurations are stored. It exposes an external interface using the OVSDB management protocol. Additionally, the `ovs-vsctl` command can be used to configure the database. Examples of such configurations include setting the SDN controller's IP Address, creating or deleting switches, adding or removing ports, and configuring the quality of service[†]. The `ovs-vswitchd` process can query the database for switch configurations using OVSDB [11].

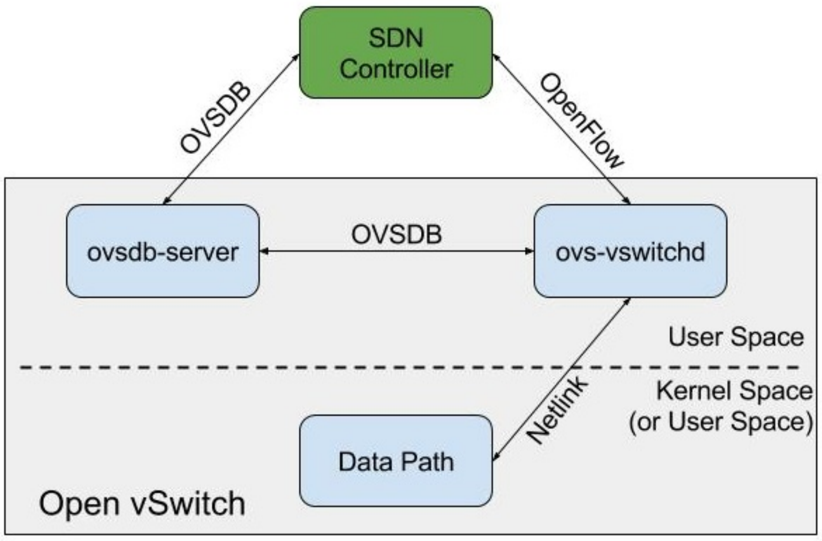


Figure 2.2: Open vSwitch Components and External Interfaces

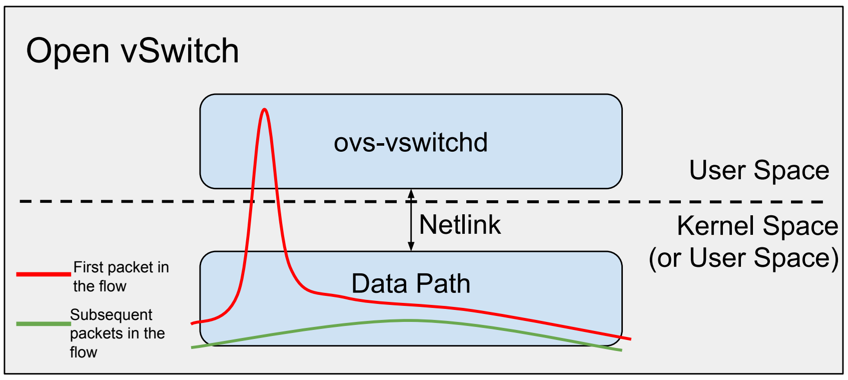


Figure 2.3: Packets flow in in Open vSwitch [13]

[†]Assigning pre-configured quality of service to a given flow is done in the flow table using OpenFlow.

2.2.1.1 Open vSwitch TLS Support

OVS can be configured to use TLS to connect to the SDN controller. The OVS documentation [14] distinguishes between two Certificate Authorities (CA): the switch's CA and the controller's CA. Every OVS instance manually generates its own private/public key pair, but there are two ways for an OVS instance to get its certificates signed:

Self signed certificates	Each OVS signs its certificate using its own private key. This avoids the need for a central authority.
Switch CA	A central authority (such as the switch's CA) signs the switch's certificate. The certificate should be presented manually to the CA, and then the signed certificate needs to be manually provisioned to the OVS. The controller can validate the switch's certificate by using the switch CA's root certificate.

Both switches and controller need to have the other's CA root certificate. The switches could obtain the controller's CA root certificate either manually or using the "bootstrap" mode [14] (where the first time that the switch connects to the controller it accepts the controller's CA root certificate).

This thesis proposes and implements a new, automatic, and secure way for OVS to generate key pairs and get both a signed certificate from a single central CA (that serves both switches and controllers) as well as the central CA's certificate.

2.2.1.2 Blocking and Non-Blocking Sockets

By default, TCP sockets are set to be blocking. For a read operation, a blocking socket will cause the program to wait for the data to be available at the input buffer to read it before returning to the program that called that read operation. While for a write operation, a blocking socket will cause the calling program to wait for the output buffer to be available to write to it before returning to the program that called the write operation.

A socket can be set to be non-blocking, and in this case the calling program will not wait for the data, if there is data available at the input buffer, the read operation immediately returns with the data, otherwise it immediately returns with an error. For the write operation, if the output buffer is available it will write to it and immediately return, otherwise it immediately returns with an error [15]. By using a non-blocking socket, the calling program does not need to spawn threads to handle the socket. In addition, in case of a connection failure, the calling program does not have to wait until the socket timeout. When studying `ovs-vswitchd` code it was clear that non-blocking sockets are used.

2.3 Trusted Execution Environment (TEE)

A Trusted Execution Environment (TEE) is an architecture that provides an isolated environment where applications can run with guarantees of code integrity and data confidentiality. TEEs operate together with a Rich Execution Environment (REE) which comprises the operating systems (such as Linux and Windows) and other applications. TEEs can offer services to applications running in REE [16]. The need for TEE stems from the increasing amount of sensitive private information software applications handle [17].

An important security feature that is offered by some TEEs is Software Attestation, which is a cryptographic signature that includes a measurement that identifies the TEE content. The signature is provided by the hardware and a verifier can verify the measurement against an expected value [18].

As discussed in detail in [18], different TEE implementations differ in the amount of software and hardware that needs to be trusted in contrast with the rest of the system. This trusted subset of the software and hardware is called the Trusted Computing Base (TCB). For hardware, the TCB includes at a minimum of the CPU chip, but could extend up to the whole motherboard including DRAM and other chips. While for software the TCB could range from small containers containing only the application to the whole virtual machine and the whole set of software running on the computer including all of the firmware.

There are a number of different TEEs. ARM TrustZone [19] is a hardware based framework that enables partitioning the system software and hardware into a “secure world" (a TEE) and a non-secure “normal world" (the REE). The hardware logic ensures that secure world resources can not be accessed from the normal world. ARM processor cores with TrustZone security extensions can safely switch between the two worlds as they are execute code. However, the normal world can only invoke the secure world via a secure monitor call. This allows for high performance and removes the need to dedicate cores to the secure world [20].

The Trusted Computing Group’s Trusted Platform Module (TPM) was initially a tamper resistant computer chip that securely stores passwords, keys, and certificates needed to authenticate the hardware platform. TPM also provides software attestation as it can be used to store measurements of the software and hardware configurations to ensure that software is not tampered with and hence is trustworthy. In TPM, the TCB can cover all of the software running on the hardware including the kernel and device drivers. The TPM specification is actively being developed by Trusted Computing Group (TCG). The evolved definition of the TPM defines four different types of TPMs to cover different implementation requirements [21]:

Discrete TPM	A dedicated tamper resistant chip is used.
Integrated TPM	TPM functionality is integrated in a chip that provides other functionality.
Firmware TPM	A software implementation of the TPM running in a TEE.
Software TPM	A TPM emulator. Such an emulator is useful for prototyping and testing purposes.

Intel® Trusted Execution Technology (TXT) [22] is based on TPM, but it reduces the TCB to cover only software running inside the virtual machine, i.e., the guest operating system and applications. TXT extends Intel's Virtual Machine Extensions (VMX) environments to enable secure deployment and use of virtual machines. TXT provides a verified launch of a virtual machine by comparing measurements of the launch components to a known good source and blocks the launching of code that deviates from the known good source. TXT also provides attestation to remote users using platform based measurements .

Intel® Safe Guard Extensions (SGX) is a new set of CPU instructions that enable application developers to confine selected code and data in protected enclaves in memory. In addition, SGX deploys hardware based mechanisms to enable remote software attestation. Section 2.3.1 gives details of SGX.

2.3.1 Intel® SGX

Intel® SGX is an extension to the Intel® x86 Architecture providing a new set of CPU instructions and memory access primitives. SGX allows developers to dedicate a private area in memory (called an enclave) such that this area can only be accessed by code running inside this enclave. Access from outside the enclave is prevented - even from privileged software such as the operating system and

BIOS. SGX limits the TCB to the processor's firmware and hardware along with the software inside the enclave. This significantly reduces the effort required for attestation evaluation. As a result, SGX guarantees both the integrity and confidentiality of the code and data stored in the enclave, while providing a means to:

- Create a hardware based attestation of the enclave's content.
- Seal the enclave's content in order to store this content outside the enclave. Thereafter, only the code in the enclave can unseal this content. The sealing mechanism also supports enclave software upgrades.

SGX enables remote software attestation so that a service provider can provision its secrets remotely in the enclave [23]. SGX supports sealing and attestation for multiple enclaves running in the same platform. Both intra-platform and inter-platform attestation are supported. In the case of intra-platform attestation, enclaves create and exchange a signed structure called a report which can be verified and the report's content can used to create a secure communication channel between the enclaves. A Quoting Enclave is introduced for inter-platform attestation. The Quoting Enclave verifies the report from the application enclave using intra-platform attestation and signs it with a device specific key before presenting it to the service provider [23] (as shown in Figure 2.4). The service provider contacts an external attestation service to verify the quote's signature. The Intel® Attestation Service (IAS) provides an Application Programming Interface (API) that service providers can use to verify enclaves' quotes [24].

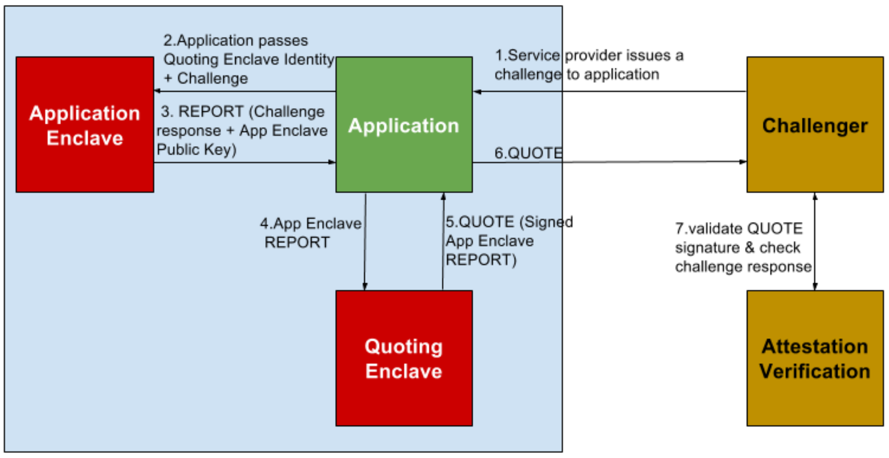

Application Enclave
2.Application passes Quoting Enclave Identity + Challenge
3. REPORT (Challenge response + App Enclave Public Key)
Application
1.Service provider issues a challenge to application
6.QUOTE
Challenger
4.App Enclave REPORT
5.QUOTE (Signed App Enclave REPORT)
7.validate QUOTE signature & check challenge response
Quoting Enclave
Attestation Verification

Figure 2.4: Inter Platform Attestation in SGX [23]

The Intel® SGX programming model differentiates between untrusted and trusted parts of the application. The trusted part executes inside the enclave, while the untrusted part, which has no access to the enclave, is responsible for creation and destruction of the enclave using SGX instructions.

SGX was introduced with the 6th Generation Intel Core processor platforms [25]. There is support for both Linux [26] and Windows [25] as well as a Software Development Kit (SDK).

2.4 Linux Integrity Measurement Architecture

The Linux Integrity Measurement Architecture (IMA) is an open source component of the Integrity Architecture based on TCG's open standards. IMA provides a means to detect if files have been changed (either deliberately or not) and it can run on platforms without a hardware TPM, but the presence of TPM provides a means to store and protect measurements in hardware. IMA provides the following functionalities [27]:

Collect	provides measurements of a file before accessing the file.
Store	stores the measurements in a kernel resident list. If a TPM is available, then the Store function extends the IMA Platform Configuration Register (PCR) with the measurement.
Attest	if a TPM is available, the TPM will be used to sign the IMA PCR value in order to enable remote verification of the measurement.
Appraise	validates the measurement against a known good value stored in the extended attribute of the file.
Protect	protects the file's extended attributes against off-line attacks.

IMA keeps a runtime measurement list that can be further anchored in hardware to prevent the measurements from being compromised. The default IMA policy measures all files needed to meet the requirements of TCB including libraries, executables, and files opened by root for reading. The policy can be modified in order to measure files other than those included in TCB. Linux Security Modules (LSM) can be deployed to label those files and those labels can be incorporated in the IMA policy [27].

Different LSMs can be used, such as SELinux [28] and Simplified Mandatory Access Control Kernel (SMACK). SMACK is a kernel module that can be used to define custom mandatory access control rules in a simple way [29].

The IMA appraisal functionality makes it possible to validate a measurement of a file against a known good value stored in the 'security.ima' extended attribute of that file. The validation of this attribute could be either to provide integrity using a hash based method or both integrity and authenticity using a digital signature based method [27].

2.5 Related Work

In this section we present related work on SDN security, SDN vulnerability analysis, analysis of TEEs and SGX, the use of SGX to create trustworthy applications, and different implementations of trusted TLS libraries in SGX.

S. Scott-Hayward, S. Natarajan, and S. Sezer carried out a survey of security in SDN in [30] from both research and industry perspectives, starting with SDN characteristics and then going through different security analyses and potential attack categories and solutions applicable to some of those categories. The survey extends to security enhancements, standardization, and open source efforts to address SDN security and future research directions. The survey also lists some open challenges when it comes to data leakage and data modification. Our work can be viewed as a solution to the data leakage security issue, as we focus on securing the keys and certificates of virtual switches.

In [31], N. Paladi and C. Gehrmann presented a security analysis for multi-tenant cloud infrastructure where network virtualization is SDN based. They introduced an adversary model and described the relevant attack vectors and then proposed a security requirements list for cloud infrastructures. In our work we address the requirement to ensure a secure enrollment of virtual switches with the SDN controller.

In [32], K. Thimmaraju, et al. demonstrated the importance of considering the dataplane in the threat model. An attacker with access to only one virtual machine can send a well crafted packet from the virtual machine to the virtual switch which can then compromise all the virtual switches, host operating system, and the SDN controller. In doing so, they pointed out a vulnerability in a vital part of Open vSwitch.

In [18], V. Costan and S. Devadas provide comprehensive background information about the Intel® Architecture, cryptographic primitives and constructs, software attestation, and physical and software attacks that are needed to understand SGX. They also described different TEEs before explaining in detail the SGX's programming model using information from Intel® manuals and inferring information from patents while pointing out some security gaps when it comes to side channel attacks.

In [33] JP Aumasson and Luis Merino presented undocumented aspects of SGX and explored possible implementation details. They also discussed possible areas in the implementation or operation that can be exploited.

Three examples of trustworthy applications created using SGX were presented in [34]. In this paper, M. Hoekstra, et al. described the design, implementation, and how trustworthiness is achieved by SGX for a one-time password generator, a secure video chat application, and Enterprise Rights Management protected applications. In [35], VC3 was introduced as an SGX based system for distributed MapReduce computations in an untrusted cloud in order to guarantee confidentiality, integrity, and correctness of code, data, and computations' results. In VC3, the TCB excludes Hadoop, operating system, and the hypervisor. VC3 demonstrated good performance with minimal overhead despite providing these security guarantees. Our work can be viewed as another example of an SGX application.

In [36], N. Paladi and C. Gehrmann presented TruSDN, a framework for bootstrapping trust in an SDN infrastructure implemented using SGX. This framework supports secure provisioning of switches in SGX enclaves, a secure communication channel between switches and SDN controller, and secure communication between endpoints in the network using session keys that are generated per flow and used only during the lifetime of the flow. The framework was implemented using OpenSGX [37], an emulator for SGX, due to unavailability of software and hardware supporting SGX. In contrast, our work is implemented and tested in actual hardware that supports SGX. Additionally, in our work we reduce the size of TCB by only confining the TLSonSGX library, generated keys and certificates, and TLS session information to the enclave, while Paladi and Gehrmann implement the whole virtual switch inside the enclave.

SecureCloud [38] is a project that targets big data applications in the smart grid domain. It provides a framework that orchestrates and schedules microservicebased applications while guaranteeing both their security and availability in untrusted cloud infrastructures by leveraging Intel SGX. The SecureCloud project assumes an adversary that can have physical access or super user privileges. Furthermore, SecureCloud enhances Intel SGX Remote Attestation by introducing a Local Attestation Service in the cloud platform. The Local Attestation Service introduces a software root of trust enclave that generates quotes which can be verified directly by the verifier without contacting IAS. Moreover, SecureCloud introduces the **LibSEAL** auditing library [39]. LibSEAL logs containers' communication and examines those logs. In addition, LibSEAL includes a part that was released later separately as an open source library under the name **TaLoS**. TaLoS [40] terminates TLS communication inside the container enclave by providing a port of LibreSSL library into SGX and thus maintaining OpenSSL API, including APIs to set private keys and certificates from outside the enclave. In this thesis projects, keys and certificates are maintained inside the enclave and no APIs are exposed to manipulate them. TaLoS was released in March 2017 and thus it was not available as an option when this thesis project started in February 2017.

At the time when the development for this thesis project started, mbed TLS [41] provided the only available port of a TLS library into SGX in Linux. Intel® [42] and wolfSSL [43] provided a port to Linux in May 2017 and June 2017 respectively. However, none of these three provided an unmodified OpenSSL API that is exposed outside the enclave. Moreover, Intel® SGX SSL library supports only a subset of OpenSSL APIs (to be used only within the enclave and not exposed outside the enclave). This subset does not cover the functionality needed by this thesis project. As a result this thesis project implements a wrapper around mbed TLS Trusted SGX library that exposes the OpenSSL APIs (that are needed for Open vSwitch TLS operations) outside the enclave as explained further in Section 5.2.3.

3 Security Model

In this chapter, we describe the adversary model together with the assumptions that have been made about the trusted elements, followed by a threat analysis for Open vSwitch, specifically identifying some attack vectors against Open vSwitch in a cloud infrastructure.

The security model excludes the security of the traffic between virtual machines or containers deployed in the cloud infrastructure - as this type of security is assumed to be handled at the application level rather than being handled at the infrastructure level.

3.1 Adversary Model

In a cloud infrastructure, virtual switches are deployed along with an SDN controller. A virtual switch is deployed in every physical compute server [44]. The adversary model assumes physical integrity and security of the hardware, including servers, storage, and networking equipment (such as hardware routers and switches). This implies that the adversary has no physical access to the hardware. We extend our assumptions to trust the TEE implementation, the cloud orchestrator, and the certificate authority.

If an adversary can gain remote access to the host operating system with full privileges, this places the attacker in a position to execute superuser privileged commands as well as accessing and changing code and stored data, including certificates and key pairs stored as plaintext on the disk. In addition, the adversary can intercept existing connections or attempt to establish new connections with the cloud SDN controller and virtual switches. We limit our adversary model by excluding the adversary's ability to execute Denial of Service attacks and side-channel attacks [45].

3.2 Threat Analysis

We apply Microsoft's **Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege** (STRIDE) threat model [46] to analyze threats to the Open vSwitch components described in Section 2.2.1. Table 3.1 summarizes the analysis, while below we describe the threats under each STRIDE category with respect to the adversary's capabilities as described in Section 3.1:

A Spoofing and Elevation of Privilege
The adversary can impersonate the controller (i.e. to carry out spoofing) and connect to the virtual switch using OpenFlow or OVSDB. The adversary can also directly execute commands that require superuser privileges (i.e. performing an Elevation of Privilege) on the host where the virtual switch is deployed. These commands interact with ovs-switchd, ovsdb-server, or the datapath processes. As a result the adversary can execute the following attacks (among others):

- Create or delete virtual switches in order to cause disruption of traffic.
- Add or remove ports from existing switches, thus making it possible to create ports that mirror traffic to ports other than those that belong to a certain tenant's instances.
- Adjust quality of service for flows in favor of or against a certain tenant.
- Insert, change, or delete flow rules in the flow table or cached flow table leading to interception, disruption, or traffic outage.

B Tampering
Given access to the host operating system, the adversary can tamper with binaries and inject malicious code that could potentially intercept or disrupt traffic. The adversary can also tamper with stored data in the flow table, OVS database, or datapath cached flow table resulting in interception or disruption of traffic.

C Repudiation
The adversary can connect to the virtual switches using OpenFlow or OVSDB impersonating the controller and carry out attacks including adjusting the quality of service of flows in favour of one of the tenants without making this tenant accountable for the attack.

D Information Disclosure
The adversary can obtain keys and certificates stored in the host where the virtual switch is deployed. Possession of this information makes it possible for the adversary to impersonate the virtual switch and intercept its communication with the controller. This could result in the adversary communicating incorrect information to the controller in order to negatively affect the controller's global view of the network and consequently causing

the controller to make incorrect routing decisions.

E Denial of Service
The adversary model that we have selected explicitly *excludes* the attacker's ability to perform Denial of Service attacks on various Open vSwitch components. Hence, we will not consider this type of attack further.

Table 3.1: STRIDE threat analysis for Open vSwitch (Gray cells are threats the thesis project defends against)

Component	Asset /Threat	Spoofing	Tampering	Repudiation	Information Disclosure*	Denial of Service**	Elevation of Privilege
ovs-vswitchd	Process		X			X	X
	Flow Table		X			X	
ovsdb-server	Process		X			X	X
	Database		X			X	
Datapath	Process		X			X	X
	Cache Table		X			X	
Protocols	OpenFlow	X		X		X	
	OVSDB	X		X		X	
	Netlink	X		X		X	
Keys and certificates			X		X		

* Side-channel attacks can be categorized under Information Disclosure but we exclude them from our adversary model.
** The adversary model that we selected excludes Denial of Service attacks.

While the threat analysis above discussed many possible vulnerabilities, we believe that ensuring code integrity, protection of keys & certificates, and the use of secure connections between the virtual switch and controller are the first line of defense against a wide range of those attacks discussed above. As a result, the remainder of this thesis will focus on implementing defensive mechanisms for Open vSwitch against those threats highlighted in gray in Table 3.1. The same mechanisms can be also implemented in the SDN controller. While we do not implement them, we assume that these mechanisms are implemented by the SDN controller for the remainder of this thesis.

4 Methodology

This chapter provides an overview of the research method utilized in this thesis project. Section 4.1 describes the research process. Section 4.2 details the experimental design. Section 4.3 explains the planned measurements. Section 4.4 discusses the reliability and validity of data collected. Finally, Section 4.5 describes the methods used for data analysis.

4.1 Research Process

The research process started with a security analysis (described in Chapter 3) and then based upon its outcomes a set of functional requirements are defined (see Section 5.1).

The applied research method is used to implement the prototype of the new library that meets those requirements and then experimental research method is applied to investigate the performance of the system and establish a relation between packet round trip latency and packet size [47].

At the start of the thesis project, the design science method was considered, where a prototype is iteratively designed, implemented and evaluated. However, as initial design iterations (that are discussed in 5.2.1) did not qualify to the implementation level, the applied and experimental research methods deemed to be the most suitable for this thesis project.

4.2 Experimental design

This section explains the elements of the experimental design. Section 4.2.1 shows the testbed used to implement the prototype, while Section 4.2.2 lists the specifications of the software and the hardware that were deployed.

4.2.1 Testbed Design

The testbed consists of two virtual machines running on top of one physical host as shown in Figure 4.1.

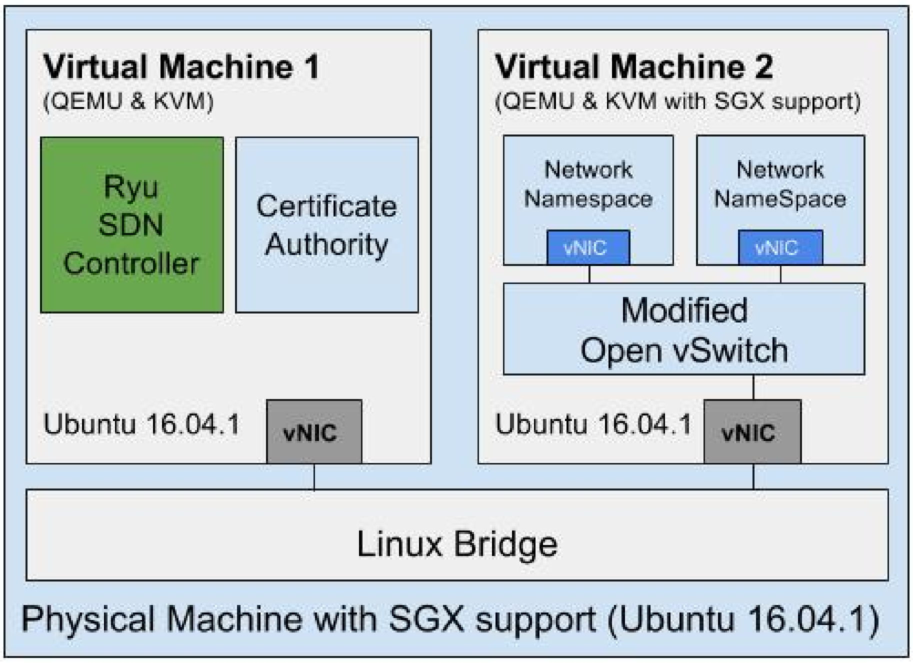


Figure 4.1: Planned Testbed (In this figure vNIC denotes a virtual NIC)

4.2.2 Hardware and Software used

The hardware and software used to construct the testbed described in Section 4.2.1 are:

- Hardware
  The hardware chosen is a Lenovo Thinkpad T460s laptop (the detailed specification is given in Table 4.1). This system's processor supports the Intel® SGX extensions required for this project.

Table 4.1: Hardware Specifications

Subsystem	**Description**
Processor	Intel® Core™ i7-6600U CPU @ 2.60GHz (Dual Core)
RAM	20 GB DDR4
Network Adapter	Intel Ethernet Connection I219-LM
Hard Disk	512GB SSD

The two virtual machines are created using QEMU and KVM with the specifications given in Table 4.2. For the second virtual machine, patched versions of QEMU and KVM provided by the SGX project are used [48]. These patched versions support the use of SGX inside the virtual machine. SGX was not used in the first virtual machine as this thesis project's focus is Open vSwitch.

Table 4.2: Virtual Machine Specifications

Subsystem	**Description**
Processor	VM_1:1 Virtual CPU, VM_2: 2 Virtual CPUs
RAM	4 GB
Hard Disk	30 GB
vNIC Driver	virtio

When configuring each virtual machine's networking, we have followed best practices stated by IBM in [49]. The laptop had hyper-threading enabled, which resulted in 4 logical CPUs. We pinned the first virtual machine to CPU 2 while we pinned the second virtual machine to CPUs 1 and 3 (same core). Within the second virtual machine, we pinned the virtual switch to one virtual CPU while we pinned the traffic generator/sink and echo server to the other virtual CPU in order to reduce inter-core communication overhead as discussed in [50]. However, due to the limited number of cores in the laptop (2 cores) we were not able to implement strict CPU isolation. (i.e. dedicate entire cores). In Section 6.2.2, we discuss the potential impact of that.

- Operating System
  For the host and virtual machines, we chose Ubuntu 16.04.1 as both Open vSwitch and SGX drivers and SDKs are supported in this specific linux distribution.

- Open vSwitch
  We chose the latest (as of 2016-12-30) Open vSwitch release 2.6.0 (commit 4b27db644a8c8e8d2640f2913cbdfa7e4b78e788).

  Inside the second virtual machine, we deployed Open vSwitch binaries that are compiled and linked with our library (as explained in Section 5.2).

Two network namespaces were created, each with a port connected to the Open vSwitch instance. This was done via executing the following linux commands:

```
# Create a new port "port1" and add it to Open
   vSwitch "ovs_switch1"
ovs-vsctl add-port ovs_switch1 port1
ovs-vsctl set Interface port1 type=internal
# Create network namespace "namespace1" and connect
   "port1" to it
ip netns add namespace1
ip link set port1 netns namespace1
# Set MAC and IP Address for "port1"
ip netns exec namespace1 ip link set port1 address
   02:00:00:00:00:01
ip netns exec namespace1 ip address add
   192.168.1.1/24 dev port1
```

The same steps were executed for the second network namespace.

- Certificate Authority
  The CA implemented (in C++) acts as a server. The CA code utilizes OpenSSL 1.1.0d for TLS communication with the virtual switch and to sign the virtual switch's and the SDN controller's certificates. We used OpenSSL instead of our new library for the CA implementation for two reasons:

  1. Our adversary model (Section 3.1) trusts the CA implementation.
  2. Ensure the interoperability between our library (deployed in the client side) and OpenSSL (deployed in the server side).

- SDN Controller
  Ryu was chosen as the SDN controller as it supports TLS communication with OpenFlow switches [51]. Additionally, this controller is easy to install and it is easy to deploy controller programs on top of it. Ryu is open source, python based, and widely used by the academic community [52] as well as in some commercial products (such as the SmartSDN controller by NTT Comware [53]).

- Intel SGX Linux Drivers and SDK
  Intel SGX Linux 1.8 release (release date 17th March 2017) is used. This release includes:

- Linux SGX driver,
- the SDK includes trusted libraries and development tools, and
- the platform software provides a runtime system library.

4.3 Planned Measurements

Besides meeting the functional requirements stated in Section 5.1, it is important to measure the performance impact of using SGX to realize TLS due to the overhead of the implementation's proposed security improvements.

4.3.1 SDN Controller Program

In practice, only the first packet in a new flow is forwarded by the virtual switch to the SDN controller and then the controller sends to the virtual switch a flow table update and the action needed to be executed by the switch to handle that first packet together with the packet itself. Subsequent packets in the flow will be handled by the virtual switch as specified using the newly installed rule in the flow table (as was explained in detail in Section 2.2.1).

In order to exercise the communications between the SDN controller and the virtual switch and to capture latency measurements, the SDN controller program in this experiment is designed to resemble a learning L2 switch, with a MAC Address to port number mapping table. However, no flow table updates are sent from the SDN Controller to the virtual switch in order to measure the extra latency induced by the controller multiple times (as otherwise we would only get one measurement of this latency for each new destination). As a result the virtual switch will keep sending all the packets in the flow to the SDN controller and the controller simply sends these packets back to the virtual switch along with the action to send the packet through the corresponding port.

4.3.2 Performance Measurements

The main performance metrics we are interested in are latency and the time required to generate key pairs and to obtain a signed certificate from the CA.

When it comes to latency, the choice of traffic generators were limited to those that can provide latency measurements. In addition, such measurements require that clocks of both traffic source and sink to be synchronized (or co exist in the same host). In this thesis, the following traffic generators were investigated: qperf, pktgen, moongen, and a Click based one. Qperf [54] is a linux based tool that can be used to measure latency between two servers over both TCP and UDP. However, it provides latency measurements per the whole TCP message or UDP fragment, while we are interested in latency measurement per individual packet.

Pktgen is a traffic generator that was originally developed to run in the linux kernel near the hardware to generate packets at high speed [55]. Several improvements were later introduced in [56]. One improvement of interest is a receiver module that can measure several parameters including the latency between sender and receiver. However, in [56] it states that two CPUs at a minimum are recommended (one for sender and one for receiver) since pktgen exhausts all the CPU resources as it sends packets. As the testbed has limited number of CPUs, pktgen can not efficiently be used.

Moongen is a DPDK based packet generator with the ability to provide accurate timestamps for latency measurements by utilizing hardware support on NICs for Precision Time Protocol (PTP) [57]. However, the vNICs used inside the network namespaces in the testbed do not support PTP.

The Click Modular Router [58] is a packet processing framework that is widely used in academia. Click can be used to implement networking elements in software, such as switches, routers, and firewalls. Click provides a modular set of elements that can be integrated to realize a specific networking application. Each element executes a simple packet processing operation, for example stripping headers or classify packets. We implemented a traffic generator and sink using the Click Modular Router that allows us to measure round trip latency for a UDP packet with different packet sizes at a rate of 500 Packets Per Second (pps) using the Click element `StoreUDPTimeSeqRecord`. By increasing the rate beyond that, much higher variance in latency results is obtained. Section 6.2.2 discusses the reasons for this behavior.

We deployed the traffic generator/sink in one network namespace and we deployed a python based, simple UDP echo server in the second network namespace. The echo server simply echoes the received UDP packet back to the traffic generator/sink. The communication between the two network namespaces is through the Open vSwitch as illustrated in Figure 4.2.

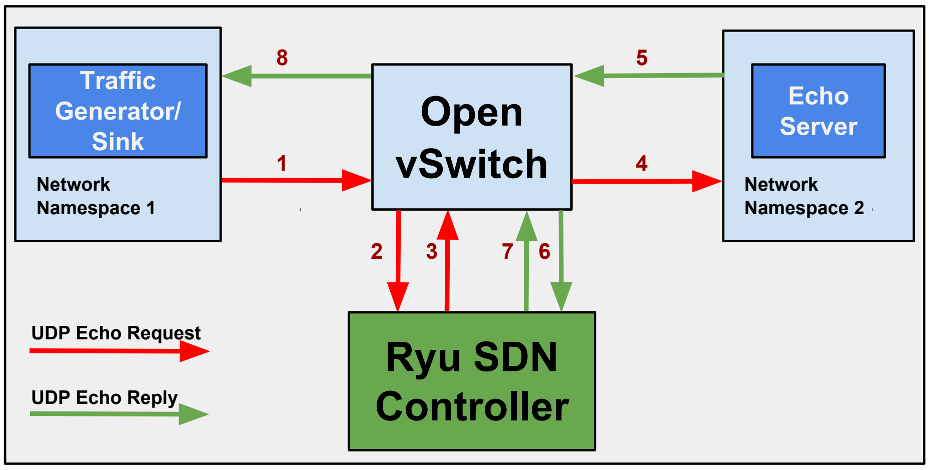


Figure 4.2: UDP Packet Path

In order to benchmark the performance, we repeated the same measurements in another virtual machine (with identical specification to virtual machine 2) using a non-patched QEMU and KVM and deploying a standard Open vSwitch that uses OpenSSL (as in a default Open vSwitch implementation).

4.4 Assessing reliability and validity of the data collected

This section discusses the reliability and validity of data collected through the measurements described in Section 4.3

4.4.1 Reliability

The reliability of the measurements comes from the fact that we have collected large numbers of samples as we discuss in Section 6.2.

4.4.2 Validity

The test cases, the traffic generator/sink together with a UDP echo server were used to collect the measurement data described in Section 4.3. The collected data were are stored in the project's repository and hence can be accessible to others. Additionally, the programs are also available and could be run by others on different or the same hardware. This ensures the validity of the results in terms of integrity and credibility.

4.5 Planned Data Analysis

This section describes the data analysis technique and software tools used.

4.5.1 Data Analysis Technique

The measurements collected are used to derive statistical measure and to plot boxplots to visualize the latency overhead introduced by the security improvements made to Open vSwitch.

4.5.2 Software Tools

R [59] is a programming language and a tool for statistical computing. We used it to compute mean, median, and first and third quartile values of the data we collected as well as to carry out linear regression analysis and to plot graphs.

5 Design and Implementation

In this chapter, we start in Section 5.1 by stating the functional requirements that the project aims to deliver. In Section 5.2, we explain in detail the design of the system.

5.1 Functional Requirements

This project aims to deliver four functional requirements (stated in Table 5.1). These requirements define what the system should do.

Table 5.1: Functional Requirements

	Functional Requirement	**Description**
1	The system should ensure that only *verified* virtual switches can access and use their private keys.	A virtual switch instance is *verified* if the current measurements of its binaries match known good measurements. As the adversary can have access to the host operating system, the keys should never be stored in plaintext on the hard disk. Instead, the keys are confined to a memory enclave, and they should be inaccessible from outside the enclave.
2	The SDN controller should be able to establish a secure channel to the virtual switches to exchange certificates.	Mutual authentication is required. Both the controller and virtual switch need to ensure the authenticity of each other to prevent an attacker from impersonating either side.
3	The system should ensure the integrity of each virtual switch and the controller's certificates.	These assurance are necessary to prevent the attacker from tampering with the certificate or impersonating the virtual switches or the controller.
4	The system should detect any modifications to virtual switch's binary.	Malicious switches (i.e. instances whose current measurements of their binaries do not match known good measurements) should be unable to enroll in the network and once detected they should be evicted from the network.

5.2 System Design

In a cloud infrastructure, virtual switch instances are created when setting up the infrastructure. During operation, when virtual machines are created or removed, ports are added or deleted from the virtual switch instance running on the underlying computing resource so that the virtual machine can communicate. Therefore, in order to meet the functional requirements described in Section 5.1 we must ensure both a secure installation of software/hardware and the initial configuration of the Open vSwitch instances in the cloud infrastructure. Securing the initial configuration centers on secure creation of keys and provisioning of certificates. Furthermore, it is important to ensure that running Open vSwitch instances are secure by ensuring the integrity of their binaries and their configurations.

The system design is based upon introducing a new library, **TLSonSGX**, to Open vSwitch to replace the existing OpenSSL library. We implemented this new library around an SGX enclave in order to ensure both code integrity and data confidentiality by leveraging features provided by Intel SGX architecture and Linux IMA.

In this design, TLS sessions originate and terminate within the enclave and the generated keys and certificates are confined within the SGX enclave, thus we can ensure their confidentiality and integrity *within* the memory enclave.

On an infrastructure level, we implement a central CA to sign both the virtual switch and the SDN controller certificates. The CA's certificate is securely and automatically provisioned to each virtual switch and the SDN controller and will be subsequently used by each of them to validate each other's certificate. In addition, the central CA validates the virtual switches before signing their certificates.

5.2.1 Design Iterations

The design and implementation have gone through several iterations focused on the following points until we reached the design described in detail in the rest of this chapter.

- **Enclave Design**
  Initially we looked into having three enclaves, an enclave to store information, an enclave to handle TLS connections, and an enclave to interact with Linux IMA and provide verdicts locally to the other enclaves. However, with such a

design there is a huge overhead imposed as the enclaves would be required to attest to each other before being able to establish secure channels to exchange information between each other. Since, the size of code and data we store inside those enclaves is much less than the Enclave Page Cache (EPC) (which is configured to the default value of 32 Megabytes), we decided to use only one enclave.

- **Interaction with IMA**
  We looked into how to get and store the known good IMA measurements. The initial design proposed stored the known good measurements inside the enclave and then exchanged the enclave quote with the CA. However, we found that it was infeasible to pre-obtain such measurements since the code we want to measure depends on the measurement's value itself and thus we decided to store the known good measurements in the CA.

5.2.2 Trusted TLS Library

SGX enclaves have limited memory and strong constraints on which instructions can be executed inside the enclave. 'Trusted libraries' are static libraries designed to be linked in an SGX enclave. As the code has to be measured when loaded into the enclave, dynamic libraries are not allowed to be linked in an enclave [60]. As discussed in Section 2.5, this thesis project uses the **mbed TLS** Trusted SGX library. Within this project, we used the commit (0ff0f8217f10a34754638a328fe02bd08c16e878) [41].

5.2.3 TLSonSGX Library Design

Following the SGX programming model, the untrusted code portion of the library is basically a wrapper that maps OpenSSL external methods (that are used by Open vSwitch) internally into enclave calls (ECALLs). The trusted portion of the code, that is contained within the SGX enclave, implements the ECALLs by utilizing the SGX trusted TLS library, specifically mbed TLS.

Since keys and certificates are confined within the enclave, Open vSwitch was modified to use only a limited set of OpenSSL external methods. These external methods are implemented in the TLSonSGX library as explained in Tables 5.2, 5.3, 5.5, and 5.6. The OpenSSL library implements three data structures: `SSL_METHOD`, `SSL_CTX`, and `SSL`. These structures are used as follows in order to establish a TLS connection:

- An `SSL_METHOD` object is created. In this object the protocol major/minor version and whether the endpoint is client or server are set.
- Then, an `SSL_CTX` object is created containing the keys, certificates, and `SSL_METHOD`.
- Finally, an `SSL` object is created to associate an `SSL_CTX` with a network connection using a socket descriptor.

Since these objects contain information that is vital for the TLS connection security, we create and confine them within the enclave. This means that although these structures are passed by Open vSwitch via the external methods we implement (so as to maintain the same interface from Open vSwitch's perspective), they are basically discarded and not passed to ECALLs but rather created, confined, and handled inside the enclave during the operation of Open vSwitch.

There is not a one-to-one mapping in mbed TLS for these three structures, hence these structures are redefined using mbed TLS primitives (specifically the `mbedtls_ssl_config` and `mbedtls_ssl_context` data structures).

Table 5.2: TLSonSGX: Library Initialization Methods

External Method	**ECALL**	**Description**
`int SSL_library_init()`	`public void ecall_ssl_library_init([in, size=buf_len] char* buf, size_t buf_len)`	This method is called once when `ovs-vswitch` is started. Once it is called, it passes the current IMA measurement to the ECALL as input. The ecall generates key pairs inside the enclave, sends the IMA measurement and certificate request to the CA, and if successful receives back a signed certificate. This method also initializes a random number generator that is required by mbed TLS.
`void SSL_load_error_strings()`	`public void ecall_ssl_load_error_strings()`	This method is called once when `ovs-vswitchd` is started. Currently this method is not implemented but rather reserved for future implementation to enable better error handling.

Table 5.3: TLSonSGX: SSL_CTX Operation Methods - Part 1

External Method	**ECALL**	**Description**
`SSL_CTX *SSL_CTX_new (SSL_METHOD* ssl_method)`	`public void ecall_ssl_ ctx_new()`	This method creates an SSL_CTX object inside the enclave setting the SSL_METHOD endpoint to client and then assigning the generated keys pair, the signed certificate, and the CA certificate to the newly created SSL_CTX object.
`long SSL_CTX_set_mode (SSL_CTX *ctx, long mode)`		Open vSwitch sets SSL_MODE_ENABLE_ PARTIAL_WRITE and SSL_MODE_ACCEPT_ MOVING_WRITE_BUFFER. However, mbed TLS supports partial writes by default and thus this method does nothing (it simply returns the mode passed into it).
`void SSL_CTX_ set_verify(SSL_CTX *ctx, int mode, void* reserved)`	`public void ecall_ ssl_ctx_ set_verify()`	This method sets the authentication mode (i.e. indicating whether verifying the peer is required or optional).
`long SSL_CTX_set_ session_cache_mode (SSL_CTX *ctx, long mode)`		Open vSwitch uses this method to disable session caching. In mbed TLS, by default there is no session resumption, thus this method does nothing and simply returns the passed mode.
`int SSL_CTX_set_ cipher_list (SSL_CTX *ctx, const char *str)`		Open vSwitch uses this method to set cipher suite negotiation in order to start from strongest and prevents uses of null encryption and MD5 for integrity checking. NULL-MD5 and RC4-MD5 are the only cipher suites supported by mbed TLS with null encryption and MD5 [61] and both are disabled by default. In addition, mbed TLS by default starts from the strongest cipher suite [62]. Therefore, this method does nothing and simply returns 1.
`long SSL_CTX_set_ options(SSL_CTX *ctx, long options)`		Open vSwitch uses this method to disable SSLv2 and SSLv3. In contrast, mbed TLS never implemented SSLv2, while SSLv3 is disabled by default. Therefore, this method does nothing and basically returns the passed options.
`void SSL_CTX_free (SSL_CTX *ctx)`	`public void ecall_ssl_ctx_ free()`	This method frees the SSL_CTX object inside the enclave

Table 5.4: TLSonSGX: SSL_CTX Operation Methods - Part 2

External Method	**ECALL**	**Description**
`int SSL_CTX_check_private_key(const SSL_CTX *ctx)`		Since the private key is confined within the enclave, this method does nothing and always returns 1.

Table 5.5: TLSonSGX: TLS Connection Setup and Shutdown Methods

External Method	**ECALL**	**Description**
`SSL *SSL_new (SSL_CTX *ctx)`	`public void ecall_ssl_new()`	This method initializes a new SSL object inside the enclave, and associates it with the previously created SSL_CTX object.
`int SSL_set_fd (SSL *ssl, int fd)`	`public int ecall_ssl_set_fd(int fd)`	This method associates a socket descriptor with the SSL object by setting the callbacks for network write and read *without* setting timeouts. Note that timeouts are unnecessary since Open vSwitch uses a non-blocking socket. This method returns 1 upon success.
`int SSL_connect (SSL *ssl)`	`public int ecall_ssl_connect()`	This method establishes the TLS handshake. Upon success the mbed TLS handshake method returns 0 but the ecall returns 1 instead as expected by Open vSwitch.
`int SSL_accept (SSL *ssl)`	`public int ecall_ssl_accept()`	Since Open vSwitch always operates as a TLS client, this method does the same things as SSL_connect.
`int SSL_get_verify_mode(SSL *ssl)`		Since Open vSwitch always verifies the SDN controller's certificate, this method simply returns 1.
`int SSL_shutdown (SSL *ssl)`	`public int ecall_ssl_shutdown()`	This method ends the connection by sending a 'close notify' to the peer.
`void SSL_free (SSL *ssl)`	`public void ecall_ssl_free()`	This method frees the SSL object inside the enclave.

Table 5.6: TLSonSGX: TLS Read and Write Operations Methods

External Method	**ECALL**	**Description**
`int SSL_read (SSL *ssl, void *buf, int num)`	`public int ecall_ssl_read ([out, size=buf_len] char* buf, size_t buf_len)`	This method allocates a buffer inside the enclave, reads data from the socket and then decrypts data inside the enclave into the buffer. Later then the content of this buffer will be copied to the buffer outside the enclave. If successful, the method returns a positive value indicating the number of bytes read, otherwise it returns a negative value that represent an error code. Open vSwitch only checks whether the value is negative and calls `SSL_get_error` to get the exact error code.
`int SSL_write (SSL *ssl, const void *buf, int num)`	`public int ecall_ssl_write ([in, size=buf_len] char* buf, size_t buf_len)`	This method copies the buffer outside the enclave to a buffer inside the enclave, encrypts the data, and then writes it to the socket. If successful, the method returns a positive value indicating the number of bytes written, otherwise it returns a negative value that represent an error code. Open vSwitch only checks if the value is negative and calls `SSL_get_error` to get the exact error code.
`extern int SSL_get_state (SSL *ssl)`	`public int ecall_ssl_ get_state()`	Due to the different representation of the SSL state machine between Open vSwitch and mbed TLS, we base the implementation of this method on the fact the Open vSwitch only examines the difference in states rather than their exact values, thus we do not provide a one to one mapping of states.
`int SSL_get_error (const SSL *ssl, int ret)`	`public int ecall_ssl_ get_error (int ret)`	While mbed TLS read or write methods return negative values as error codes, OpenSSL uses `SSL_get_error` to get the error code. We store the error codes from read and write methods in the SSL object so that this method can return the latest error code.
`int SSL_want (const SSL *ssl)`		This method maps errors returned from `ecall_ssl_get_error` to either `SSL_READING` or `SSL_WRITING` indicating the non-blocking socket's state.

Finally, OpenSSL and mbed TLS implement different error code numbers, so we had implemented a one-to-one mapping of error codes so that Open vSwitch receives the same error code values from either library (OpenSSL or TLSonSGX) as shown in Table 5.7.

Table 5.7: TLSonSGX: Error Codes

OpenSSL Error Code	**mbed TLS Error Code**	**Description**
`SSL_ERROR_WANT_READ`	`MBEDTLS_ERR_ SSL_WANT_READ`	For a non-blocking socket, a read retry is needed
`SSL_ERROR_WANT_WRITE`	`MBEDTLS_ERR_ SSL_WANT_WRITE`	For a non-blocking socket, a write retry is needed
`SSL_AD_CLOSE_NOTIFY`	`MBEDTLS_ERR_ SSL_PEER_CLOSE_ NOTIFY`	Indicates that the peer has notified that the connection is going to be closed.

5.2.4 Key and Certificate Generation

In order to create the private/public key pairs and obtain switch and CA certificates, the workflow illustrated in Figure 5.1 is executed:

1. When `ovs-vswitchd` is started, it calls the `SSL_library_init` method from the statically linked TLSonSGX library.
2. `SSL_library_init` gets the current IMA measurement of `/usr/sbin/ovs-vswitchd` and passes it in the ECALL to the SGX enclave.
3. Inside the enclave, the private key, public key, and certificate request are generated.
4. The SGX enclave establishes a TLS connection with the CA.
5. Once the connection is established, the SGX enclave will receive the CA's certificate & store it and then send the IMA measurement to the CA.
6. The CA will validate the IMA measurement against a known-good measurement that it already has and sends a verdict to the enclave.

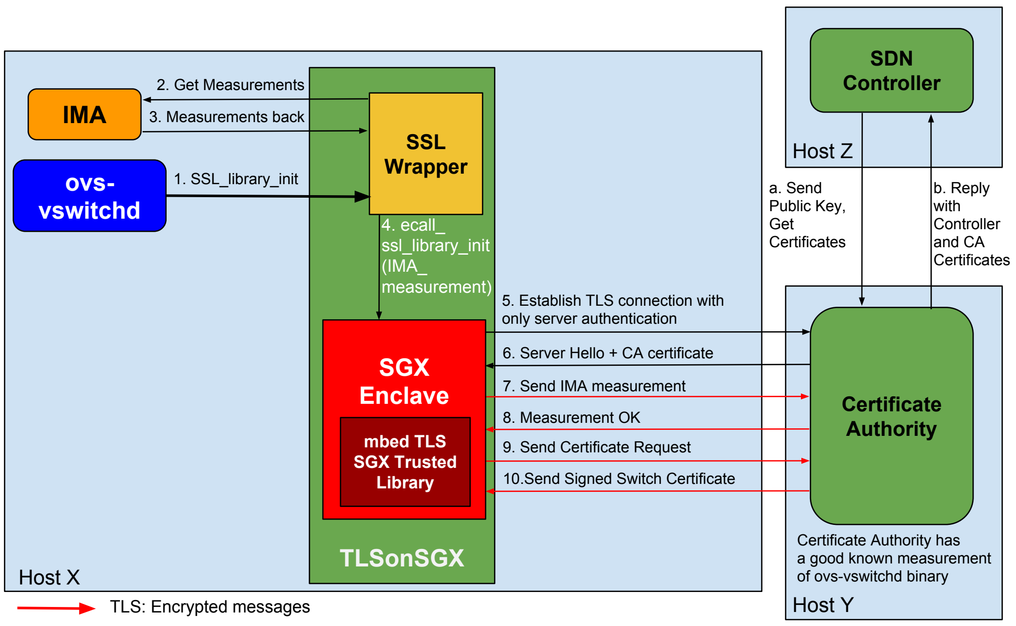
7. If the verdict is OK, then the CA will maintain the TLS session and wait for a certificate request, otherwise it aborts the connection.

8. The SGX enclave sends the certificate request over TLS to the CA.

9. The CA signs the certificate and returns it over TLS to the SGX enclave.

10. The SGX enclave stores the private key, signed switch certificate, and CA's public certificate.

It is assumed that the controller employs similar security mechanisms to create the controller's public-private key pairs and to obtain from the CA both the CA's and the controller's certificates.



SDN Controller
Host Z
a. Send Public Key, Get Certificates
b. Reply with Controller and CA Certificates
IMA
2. Get Measurements
3. Measurements back
SSL Wrapper
ovs-vswitchd
1. SSL_library_init
4. ecall_ ssl_library_init (IMA_ measurement)
SGX Enclave
mbed TLS SGX Trusted Library
TLSonSGX
5. Establish TLS connection with only server authentication
6. Server Hello + CA certificate
7. Send IMA measurement
8. Measurement OK
9. Send Certificate Request
10.Send Signed Switch Certificate
Certificate Authority
Certificate Authority has a good known measurement of ovs-vswitchd binary
Host X
Host Y
TLS: Encrypted messages

Figure 5.1: Keys and Certificates Generation

5.2.5 Labelling Binaries and Performing Runtime Measurements

In order to provide the static library with runtime measurements, the Open vSwitch installation needs to be augmented with the following extra steps:

1. Add SMACK and IMA as kernel parameters.

2. Use SMACK to label files that need to be protected from tampering (specifically `/usr/sbin/ovs-vswitchd` and `/usr/sbin/ovsdb-server`).

3. Define an IMA policy to measure those files when they are to be executed.

Once the computer hosting Open vSwitch starts, IMA measures these files and remeasures them if they are re-executed at runtime and stores the new measurements in the default file:
`/sys/kernel/security/ima/ascii_runtime_measurements`.
This file is a kernel resident list and it is read-only.

5.2.6 Switch to Controller TLS Connection Establishment

Once the key and certificates are securely provisioned in the enclave, in order for Open vSwitch to establish a TLS connection with the SDN controller to send packets and receive flow updates, the system functions as follows (as illustrated in Figure 5.2):

1. `ovs-vswitchd` is configured manually with the SDN controller's IP Address, using the command:
   ```
   ovs-vsctl set-controller <switch_name> \
   ssl:<SDN_Controller_IP>:6633†
   ```

2. `ovs-vswitchd` invokes the `SSL_connect` method from the TLSonSGX library which in turns invokes the ECALL to the SGX enclave.

3. The SGX enclave establishes a TLS connection with the SDN controller.

4. The SDN controller sends its certificate over the TLS connection.

5. The SGX enclave validates the controller's certificate using the CA's certificate stored in the enclave.

6. The SGX enclave sends the switch's certificate and the SDN controller validates the switch's certificate using the CA's certificate.

A similar flow can also be applied for the `ovsdb-server`'s communication with the SDN controller and for its communication with an instance of `ovs-vswitchd`.

†6633 is a well known port used for OpenFlow

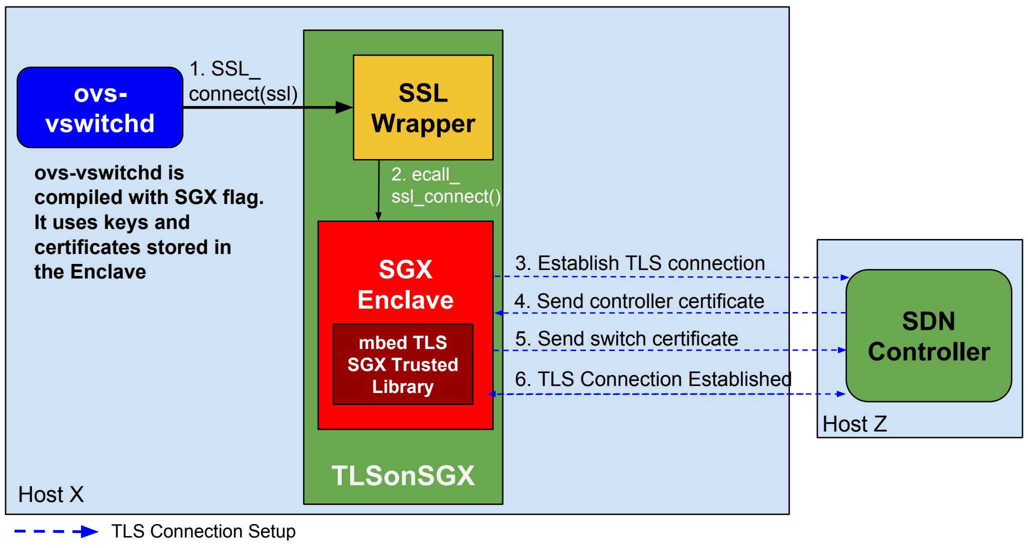

ovs-vswitchd
1. SSL_ connect(ssl)
SSL Wrapper
2. ecall_ ssl_connect()
ovs-vswitchd is compiled with SGX flag. It uses keys and certificates stored in the Enclave
SGX Enclave
mbed TLS SGX Trusted Library
3. Establish TLS connection
4. Send controller certificate
5. Send switch certificate
6. TLS Connection Established
SDN Controller
Host Z
TLSonSGX
Host X
TLS Connection Setup

Figure 5.2: Open vSwitch - SDN Controller TLS Connection Setup

5.2.7 Open vSwitch Modifications

The only file in the Open vSwitch source code repository that utilizes OpenSSL library is `stream-ssl.c`. All other files (such `ovs-vswitch` and `ovsdb-server`) simply use `stream-ssl`.

We augmented the Open vSwitch configuration script and `stream-ssl.c` with a new compilation flag `SGX`. When Open vSwitch is compiled with this flag set, our static library, **TLSonSGX**, will be used instead of the OpenSSL library in `stream-ssl.c`. Moreover, the sections of `stream-ssl.c` that load keys and certificates from the file system are omitted as they are unnecessary.

6 Results and Analysis

This chapter presents and analyzes the results from both functional tests (Section 6.1) and performance tests (Section 6.2). Section 6.3 discusses our compliance with the functional requirements, while Section 6.4 concludes this chapter with a presentation of a security analysis for our implementation.

6.1 Functional Test

We tested that our implementation delivers the following Open vSwitch functionalities:

- Connects to SDN controller using TLS.
- Exchanges OpenFlow messages (as described in [63]) with an SDN controller that implements a learning switch's functionality. This includes the following:
  - Controller to Switch Messages:

    Send Features Request: Open vSwitch replies with the switch's features.

    Modify Flow Entry: add a new flow, modify an existing flow, or delete a flow.

    Packet Out: The controller instructs the switch to send a packet either by flooding it or forwarding it through a specific port.
  - Switch to Controller Messages:

    Packet In: The switch forwards the received packet to the controller.

    Flow removed: sent by the switch when a flow is deleted or timed-out.

    Port Status: sent by the switch when a port is added, deleted, or modified.

6.2 Performance Measurements Results

We carried out the measurements described in Section 4.3.2. In Section 6.2.1 we discuss the measurements of the time needed to generate keys and certificates, while in Section 6.2.2 we present round trip latency measurements for a UDP packet between a client and an echo server.

6.2.1 Keys and Certificate Generation Time

This timing concerns the time from when the Open vSwitch invokes the `SSL_library_init` method to execute the steps described in Section 5.2.4 and ends when the key pairs and signed certificate are loaded into the enclave's memory.

As it is easy to automate the measurement, 1000 readings were deemed to be enough to get a statistical power of 0.99 with a 95% significance level assuming a normal distribution of measurements and a small effect size (0.2) as per the power analysis below using R. Results of the measurements are shown in Table 6.1.

```
>library(pwr)
>pwr.t.test(n=1000,d=0.2,sig.level=0.05,power=NULL,
+type="one.sample")

    One-sample t test power calculation

             n = 1000
             d = 0.2
     sig.level = 0.05
         power = 0.9999935
   alternative = two.sided
```

Table 6.1: Keys and Certificate Generation Time

Number of measurements	1000 readings
Mean	0.344 seconds
Variance	0.0488
1st Quartile	0.186 seconds
Median	0.276 seconds
3rd Quartile	0.434 seconds

In a standard Open vSwitch, there is no corresponding measurement as keys and certificates are handled manually as described in Section 2.2.1.1. However, as this operation is only executed once when the `ovs-vswitchd` process starts, these measurements indicate that there is a little overheard (in practice) introduced by this implementation.

6.2.2 Packet Round Trip Latency

This section discusses the packet round trip latency measurements for the setup described in Section 4.3.2 and analyzes different aspects related to these results. The measured time does neither include the time needed to generate keys (already measured in Section 6.2.1) nor the TLS session establishment time as the TLS session is already established before packets can flow. The TLS session remains up unless one of the two ends (the Open vSwitch or the SDN controller) terminates the session.

6.2.2.1 Packet Size The largest IP packet size received by the Open vSwitch from the traffic generator is bounded by the Maximum Transmission Unit (MTU) of the network namespace port connected to the Open vSwitch (in this case the MTU was 1500 bytes). Open vSwitch encapsulates the whole received packet (including the Ethernet header) in an OpenFlow **Packet In** message, adding an 18 bytes header [3], which in return is encapsulated in a TLS record that is sent from the Open vSwitch to the SDN controller. If the packet sent by the traffic generator is larger than the MTU, then it will be fragmented and Open vSwitch will handle it as two separate **Packet In** messages to the SDN controller.

The TLS record basically adds a 5 byte header. Depending on the cipher suite negotiated between server and client, a padding field (up to 15 bytes) is added if a block cipher is used and the TLS record is appended with a Message Authentication Code (MAC) field that is computed over the data. As discussed in Table 5.3 Open vSwitch is configured to negotiate the strongest cipher suite while it prevents null encryption. When looking into handshake messages exchanged between Open vSwitch and the SDN controller during testing, the negotiated cipher suite is **ECDHE-RSA-AES256-SHA**. This means that:

- For key exchange, Elliptic Curve Ephemeral Diffie Hellman (ECDHE) is used to compute the ephemeral TLS premaster secret (that is used later to generate the TLS master secret) and Rivest–Shamir–Adleman (RSA) signatures are used to authenticate the secret [64].
- Advanced Encryption Standard (AES) with 256 bits key length is used for bulk encryption in Cipher Block Chaining (CBC) mode with Secure Hash Algorithm 1 (SHA-1) for MAC construction [65]. SHA-1 creates a 20 bytes MAC that is appended to the end of the TLS record.

We measure the latency for increasing packet sizes (adding 64 bytes in every step) ranging from 64 bytes up to 1408 bytes, including the Ethernet and IP headers (minus the Cyclic Redundancy Check (CRC)) to avoid subsequent fragmentation between the Open vSwitch and the SDN controller.

We repeated the measurement 10,000 times for every packet size. With such a large sample size, a normal distribution can be assumed and a statistical power of 1 can be reached with a significance level of 95% and a small effect size (0.2) as per the power analysis below done using R.

```
>library(pwr)
>pwr.t.test(n=10000,d=0.2,sig.level=0.05,power=NULL,
+type="one.sample")

    One-sample t test power calculation

              n = 10000
              d = 0.2
      sig.level = 0.05
          power = 1
    alternative = two.sided
```

6.2.2.2 Packet Rate Selection and CPU Utilization

From the data we captured, we excluded outliers whose round trip latency exceeds 2.5 milliseconds, since they make the boxplot less readable. When testing OpenSSL there were 5237 outliers, while for TLSonSGX there were 11622 outliers out of 220000 samples for each implementation. When analyzing the reason for these outliers and why the number differs between the two implementations, the CPU utilization was investigated. In both implementations, inside the virtual machine, the first vCPU reaches 100% utilization due to the Click packet generation process that is pinned to it, even with lower rates than 500 pps (i.e., 50 , 100 , 200 pps). However, the second vCPU, where `ovs-vswitchd` process is pinned, has a higher average CPU utilization when TLSonSGX is used compared to OpenSSL (as stated in Table 6.2). When increasing the rate beyond 500 pps, then the second vCPU's utilization increases along with the average latency.

Table 6.2: Packet Rate vs. Average CPU Utilization

Packet Rate	OpenSSL	TLSonSGX
500 pps	25%	61%
1000 pps	40%	78%
2000 pps	49%	96%

Therefore, 500 pps was picked as a suitable and optimal maximum rate and further measurements and analysis was based on this rate. The increased CPU utilization is expected when SGX is used due to overhead of memory transition to and from the memory enclave.

6.2.2.3 Latency and Packet Size The packet round trip latency measurements results are plotted using a boxplot to compare our implementation using **TLSonSGX** with the standard Open vSwitch with OpenSSL when forwarding UDP packets of a range of sizes. Outliers were excluded as explained in the previous section. Figure 6.1 shows a plot of latency versus packet size.

Each box represents the data between first and third quartile, with the thick line in the box representing the median. The upper whisker is equal to the minimum value between the largest value in the data and 3rd Quartile + 1.5*IQR, where IQR is the interquartile range. The lower whisker is the maximum value between the smallest value in the data and 1st Quartile - 1.5*IQR [66].

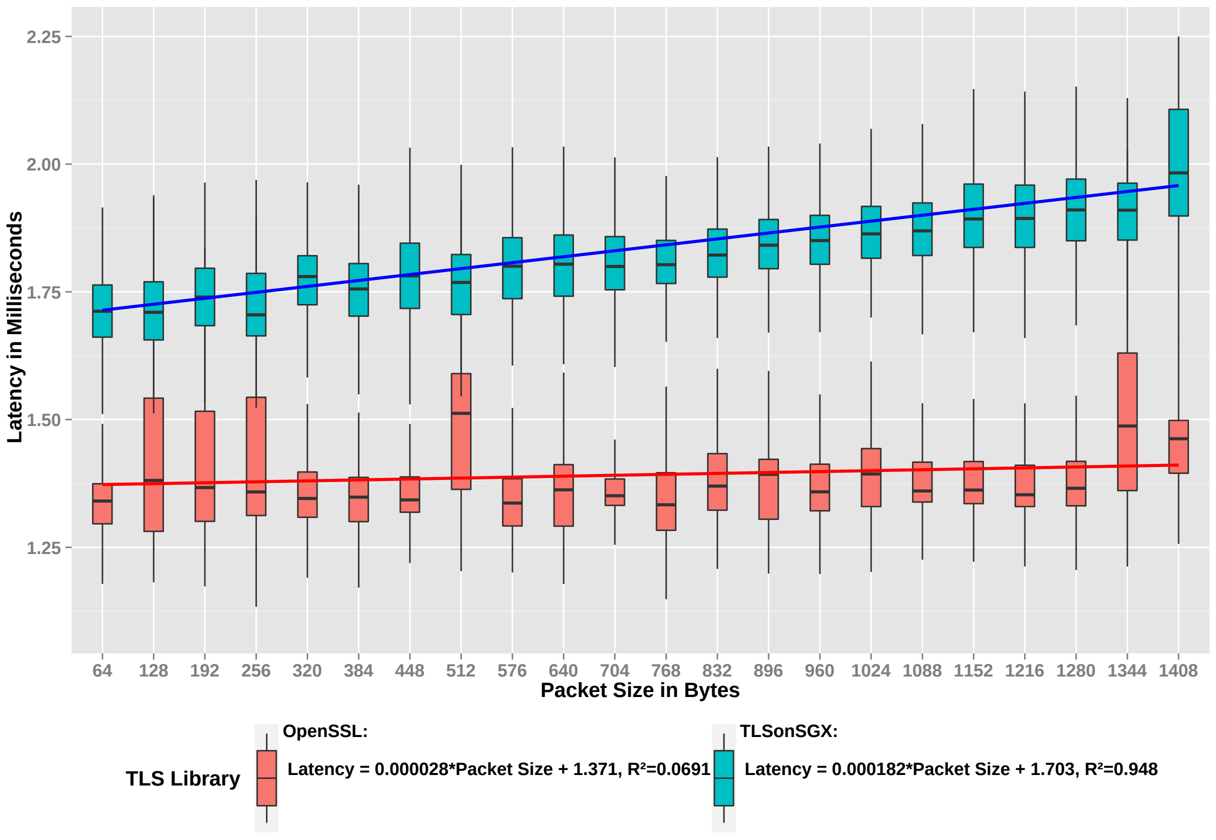


Figure 6.1: UDP Packet Round Trip Latency vs. Packet Size

In addition, a linear regression analysis for the mean values was carried out. From this analysis, it can be observed that at zero byte TLSonSGX has an extra cost of 0.33 milliseconds compared to OpenSSL. Then for both implementations the latency increases almost linearly as packet size increases, this increase is estimated to be:

OpenSSL: 28 nanoseconds per byte.

TLSonSGX: 182 nanoseconds per byte.

The linear increase is expected as more time is needed to process larger packets. However, the increase per byte is higher in TLSonSGX compared to OpenSSL (154 nanoseconds per byte). This, along with the extra cost of 0.33 milliseconds at zero byte, are also expected due to the overhead of transitions to and from the memory enclave. In order to analyze and break down the time difference between OpenSSL and TLSonSGX, we traced how many ECALLs are indirectly called by `ovs-vswitchd` during a packet's round trip. Their numbers are as follows:

- Once a packet is received at an Open vSwitch port from the network name space, `ovs-vswitchd` triggers `ecall_ssl_write` to encrypt and send the packet to the SDN controller, while checking the SSL state (`ecall_ssl_get_state`) before and after the write ECALL.
- Since, as discussed in Section 2.2.1.2, `ovs-vswitchd` uses non-blocking sockets, `ovs-vswitchd` keeps reading and returning from the socket (`ecall_ssl_read`), while comparing the SSL state before and after the read (`ecall_ssl_get_state`).
- If a negative value is returned (WANT_READ) from `ecall_ssl_read` then it triggers (`ecall_ssl_get_error`) to retrieve the error code which indicates that the read call must be repeated and accordingly continue the loop.
- If a positive value is returned, this means that there is a response from the controller. The controller will respond with two packets, the original packet itself and the action needed by the switch to forward the packet to the second network name space.
- The exact same flow will happen during the packet's return trip from the second network name space to the first one.

We measured the time consumed for each ECALL and we repeated the measurement 10000 times per packet size. With such a large sample size, a normal distribution can be assumed and a statistical power of 1 can be reached as per the power analysis done in Section 6.2.2.1. Table 6.3 lists the mean values for each ECALL. The last column in the table is the sum of all ECALLs times per packet round trip as explained above.

We noticed also that `ecall_ssl_write` takes longer (and increases with packet size) than the other ECALLs. This is because `ecall_ssl_write` is the only ECALL that writes from a buffer with a pointer outside the enclave (unprotected memory) to the enclave memory. All the other ECALLs do the opposite. According to [67], ECALLs that passes an external pointer into the enclave are slow since a buffer is allocated inside the enclave memory and the content and the size of the buffer referenced by the external pointer are verified for every call to prevent overwriting enclave code or data before copying the contents of the external buffer into the enclave memory.

As a result, in a typical deployment where only the first packet in the flow is sent to the SDN controller, if this packet can be crafted to be small enough (64 bytes), then the latency can be optimized and hence the time needed to add the flow rule in the Open vSwitch flow table (that subsequent packets in the flow will match).

Table 6.3: Analysis of packet latency (all measurements are in milliseconds**)

Packet Size (Bytes)	TLS on SGX	OpenSSL	Difference	ecall_ ssl_ read	ecall_ ssl_ write	ecall_ ssl_ get_ state*	ecall_ ssl_ get_ error*	Total enclave access time
64	1.6500	1.2682	0.3817	0.0047	0.0646	0.0047	0.0043	0.2966
128	1.6667	1.2722	0.3944	0.0048	0.0676	0.0047	0.0043	0.3040
256	1.6820	1.2844	0.3976	0.0049	0.0725	0.0047	0.0043	0.3146
512	1.6852	1.2955	0.3897	0.0049	0.0828	0.0047	0.0043	0.3350
1024	1.6963	1.3145	0.3818	0.0049	0.1022	0.0047	0.0043	0.3740

* `ecall_ssl_get_state` and `ecall_ssl_get_error` are independent of packet size
. ** Those measurements were captured in a different iteration than the one depicted in Figure 6.1.

6.3 Compliance with Functional Requirements

In this section, we discuss our implementation's compliance with the functional requirements stated in Section 5.1. Table 6.4 presents each requirement and examines how the implementation is compliant with respect to it.

Table 6.4: Compliance with Functional Requirements

	Functional Requirement	**Compliance**
1	The system should ensure that only *verified* virtual switches can access and use their private keys.	The CA verifies the IMA measurement of `ovs-vswitchd` *before* signing the certificate that in turn is stored in the enclave. Unless the CA verifies the IMA measurement the TLS connection to the controller can not be established. Moreover, the keys are generated and used only within the enclave.
2	The SDN controller should be able to establish a secure channel to the virtual switches to exchange certificates.	With a central CA, both the controller and virtual switch can mutually validate each other's certificate using the same CA's certificate.
3	The system should ensure the integrity of each virtual switch and the controller's certificates.	Keys and certificates are stored and used only within the enclave.
4	The system should detect any modifications to virtual switch's binary.	Malicious switches will never receive a signed certificate, as the CA will detect their invalid IMA measurements. As a result, malicious switches can not establish a connection with the SDN Controller.

6.4 Security Analysis

In this section, we explain how our implementation mitigates the threats we analyzed in Section 3.2.

1. Spoofing and Repudiation
   By using TLS for the TCP connection between the SDN controller and

the virtual switch together with the introduction of a centralized CA, the implementation protects against such threats as the attacker can not impersonate the virtual switch without getting hold of the virtual switch private key which never leaves the enclave. Furthermore, both the SDN controller and the virtual switch can mutually authenticate and validate each other's certificates using the CA certificate.

2. Tampering
   As binaries are measured using IMA and measurements are verified by the CA, our implementation protects against an adversary tampering with the binaries (specifically `ovs-vswitchd` and `ovsdb-server`). Tampering with the flow table (stored in unprotected memory) or OVS database content (stored as a file in the disk) are not addressed by our implementation.

3. Information Disclosure
   The implementation prevents from the disclosure of keys, certificates and TLS session information as they are securely stored in the enclave, hence they are not available in plain text outside of the CA. Furthermore, TLS1.2 has addressed flaws that previously enabled known and chosen plaintext attacks [68]. On top of that, as discussed in Table 5.3, Open vSwitch implementation ensures that the strongest cipher suite is negotiated.

4. Elevation of Privilege and Denial of Service
   The implementation did not address Elevation of Privilege, but such a threat can be mitigated by disabling command line interfaces that can be used to manipulate the flow table or the Open vSwitch database, such as `ovs-vsctl` and `ovs-ofctl`. Denial of Service attacks were excluded from our adversary model.

7 Conclusion and Future Work

This chapter concludes this Master's thesis. Section 7.1 explains the limitations of this project. Section 7.2 discusses potential future work within the field of this project. While Section 7.3 presents our reflections on different aspects of this project. Finally, Section 7.4 presents conclusions from the analysis of the results in Chapter 6.

7.1 Limitations

We implemented a prototype and tested it using one laptop (with only two cores) and used virtual machines with SGX support to host the virtual switches, the SDN controller, and network namespaces as described in Section 4.2.1. While it was possible to demonstrate the feasiblity of **TLSonSGX** and provide a relative comparison to OpenSSL, using a laptop deemed to be a bottleneck when it comes to dedicating CPU cores for processes and thus introduced limitations on traffic generator selection and the packet rate that can be used.

At the time the hardware procurement for this thesis started (November 2016), there was no available servers with SGX support. Intel® released its server system with SGX support in March 2017 [69]. Another option would have been public clouds. Google Cloud Platform announced that it supports Skylake architecture [70], but there is no mention of SGX support. Google Cloud Platform uses KVM [71]. There is no official KVM support for SGX. This thesis project uses a patched version that is not recommended to be used for production [72]. On September 14th 2017, Microsoft Azure introduced support for SGX [73] as part of an Early Access Program (limited access), this can be explored as part of future work to setup multiple machines for testing in order to assemble a more realistic deployment in a cloud environment such as the one depicted in Figure 2.1.

The Open vSwitch data path (fast path) can be deployed in either kernel space or user space. In this thesis project, only a kernel space implementation was used as we were interested in examining the connection between the controller and the virtual switch, which exercises the Open vSwitch slow path, rather than examining the virtual switch's throughput which exercises the fast path (fast and slow path are explained in Section 2.2.1).

We have not implemented a complete mapping of all OpenSSL methods in the TLSonSGX library. Instead, we mapped only the methods needed for Open vSwitch to operate while it is integrated with the SGX enclave as described in the design of TLSonSGX library in Section 5.2.3 .

7.2 Future Work

Throughout the thesis project, many improvements and potentials for future work are identified. These include:

- As briefly suggested in Section 3.2, TLSonSGX can be linked, with minor adaptations, with the SDN controller in order to generate keys and obtain controller's certificate securely and automatically as well as terminating TLS sessions with the virtual switch in the SGX enclave.
- The current implementation supports only one virtual switch connecting multiple virtual machines per physical host as only one SSL context is created and kept inside the enclave. An improvement would be to introduce support for multiple switches per host by extending the library to support multiple SSL contexts inside the enclave while keeping track of the correct mapping between the switch and its corresponding SSL context.
- TLSonSGX can be extended to protect the flow table (stored in unprotected memory) or OVS database content (stored as a file in the disk) from tampering by storing them in the SGX enclave.
- IMA measurements can be stored in the TPM. Instead of fetching the measurements from the file system, the SGX enclave could read the value directly from TPM. This adds an additional security guarantee of the measurements integrity by the hardware. However, a TPM chip might not necessarily be available in all platforms.
- Side channels attacks require a long time to analyze memory access patterns. In order to mitigate such attacks targeting the virtual switch's private key, Open vSwitch can periodically generate new keys pairs and request a new certificate from the CA.
- In order for keys and certificates to survive host reboots, the enclave could deploy sealing mechanisms to 'seal' the enclave, i.e. encrypt it, export it from the enclave, and store it on the local hard disk. We did not prioritize implementing this feature within the thesis project's timeframe, as we did not see it as an essential feature but rather a good feature to have. Moreover, the time to generate new keys and get a new certificate (as described in Section 6.2.1) is only roughly 1/3 of a second.

- The current CA verification mechanism can be extended with a SGX Remote Attestation flow, thus providing a way for the CA to ensure that Open vSwitch is running on a genuine SGX platform. The CA would need to contact the IAS either directly (which requires internet connectivity) or through a proxy.

7.3 Required Reflections

This section presents reflections regarding the environmental, ethical, and economic aspects of this project.

7.3.1 Environmental and Sustainability Aspects

Virualization technology promotes resource sharing and hence less consumption of energy. On the other hand, many vulnerabilities can result from resource sharing, hence an improvement in security while still allowing resource sharing will ensure both a more robust virtualized infrastructure and enable sharing that otherwise could not be done.

7.3.2 Ethical Aspects

Ensuring the security and isolation of tenants residing on cloud infrastructure is crucial to guarantee the privacy of those tenants.

7.3.3 Economic Aspects

The proposed improvements in security should help to reduce potential attacks and thus the cost incurred by cyber crime which is projected to increase every year [74]. In addition, The thesis introduces an automated way to generate keys and obtain certificates that replaces the manual procedure, this results in reduction in man-hours needed to setup cloud infrastructure.

7.4 Conclusion

Open vSwitch is a key component in a cloud infrastructure and data plane security is crucial to enable isolation between tenants. We introduce and implement the TLSonSGX library by the leveraging Intel® SGX architecture. The library provides a new, secure, and automatic mechanism for Open vSwitch to generate keys and obtain signed certificates while keep them secured within a memory enclave. This new mechanism is a replacement for the existing manual and unsecure procedures (as described in Open vSwitch project). TLSonSGX also confines all the TLS connections to the SDN controller within the enclave to ensure that keys, certificates, and session data are inaccessible outside the enclave.

We tested the functionality and performance of our implementation and we provided performance metrics in a comparison with a standard Open vSwitch that uses OpenSSL. Our results show that generating keys and certificates using TLSonSGX takes less than 0.5 seconds while using TLSonSGX adds 30% latency overhead for the first packet in each flow compared to using OpenSSL. This additional latency is due to enclave access time. These results show that TLSonSGX can contribute to reduce Open vSwitch TLS configuration overhead and also to enhance Open vSwitch security.

References

[1] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN: an intellectual history of programmable networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014. DOI: 10.1145/2602204.2602219 Last accessed on 2017-10-20. [Online]. Available: http://doi.acm.org/10.1145/2602204.2602219

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. DOI: 10.1145/1355734.1355746 Last accessed on 2017-10-20. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[3] “OpenFlow Switch Specification-Version 1.4.0, Open Networking Foundation,” 2013, Last accessed on 2017-05-28. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[4] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, “Feature-based comparison and selection of Software Defined Networking (SDN) controllers,” in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. IEEE, Jan 2014. DOI: 10.1109/WCCAIS.2014.6916572 pp. 1–7.

[5] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, “Extending Networking into the Virtualization Layer.” in *Hotnets*, 2009, Last accessed on 2017-10-24. [Online]. Available: http://openvswitch.org/papers/hotnets2009.pdf

[6] S. Baucke, K. Mestery, A. Shaikh, and C. Wright, “OpenDaylight: An Open Source SDN for your OpenStack Cloud,” *OpenStack Summit, Hong Kong*, 2013, Last accessed on 2017-10-24. [Online]. Available: https://www.openstack.org/assets/presentation-media/OpenDaylightOpenStackIcehouseSummit.pdf

[7] “Networking:Bridge [Linux Foundation Wiki],” Last accessed on 2016-11-01. [Online]. Available: https://wiki.linuxfoundation.org/networking/bridge

[8] “Networking - KVM,” Last accessed on 2016-11-01. [Online]. Available: http://www.linux-kvm.org/page/Networking

[9] "How to Build the Kernel module & userspace daemons for Windows," Last accessed on 2016-11-04. [Online]. Available: http://openvswitch.org/support/dist-docs/INSTALL.Windows.md.html

[10] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of Open vSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015. ISBN 978-1-931971-218 pp. 117–130, Last accessed on 2017-09-28. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789779

[11] V. Goldberg, F. Wohlfart, and D. Raumer, "Datacenter Network Virtualization in Multi-Tenant Environments," in *8. DFN-Forum Kommunikationstechnologien*, June 2015, Last accessed on 2017-05-28. [Online]. Available: http://www.net.in.tum.de/fileadmin/bibtex/publications/papers/DatacenterNetworkVirtualizationInMulti-TenantEnvironments.pdf

[12] S. Shanmugalingam, A. Ksentini, and P. Bertin, "DPDK Open vSwitch performance validation with mirroring feature," in *2016 23rd International Conference on Telecommunications (ICT)*. IEEE, 2016. DOI: 10.1109/ICT.2016.7500387 pp. 1–6.

[13] J. Pettit and E. Lopez, "OpenStack: OVS Deep Dive," *OpenStack Summit, Hong Kong*, 2013, Last accessed 2016-12-27. [Online]. Available: http://openvswitch.org/slides/OpenStack-131107.pdf

[14] "Open vSwitch with SSL," Last accessed on 2016-11-10. [Online]. Available: https://github.com/openvswitch/ovs/blob/master/INSTALL.SSL.rst

[15] "Client/server socket programs: Blocking, nonblocking, and asynchronous socket calls | IBM Knowledge Center," Last accessed on 2017-10-21. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.hala001/orgblockasyn.htm

[16] M. Mohanty, V. Do, and C. Gehrmann, "Media data protection during execution on mobile platforms–a review," SICS Swedish ICT AB, Tech. Rep., 2014, Last accessed on 2017-10-12. [Online]. Available: http://soda.swedish-ict.se/5685/1/Media_Data_Protection_during_Execution_on_Mobile_Platforms_%E2%80%93_A_Review.pdf

[17] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "Trusted execution environments on mobile devices," in *Proceedings of the 2013 ACM SIGSAC conference*

on Computer & communications security, ser. CCS ’13. New York, NY, USA: ACM, 2013. DOI: 10.1145/2508859.2516758. ISBN 978-1-4503-2477-9 pp. 1497–1498, last accessed on 2017-10-21. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516758

[18] V. Costan and S. Devadas, “Intel SGX Explained,” Cryptology ePrint Archive, Report 2016/086. https://eprint.iacr.org/2016/086, Tech. Rep., 2016, Last access on 2017-09-20.

[19] T. Alves and D. Felton, “TrustZone: Integrated hardware and software security,” *ARM white paper*, vol. 3, no. 4, pp. 18–24, 2004.

[20] “ARM Security Technology: Building a Secure System using TrustZone® Technology,” *ARM Technical White Paper*, 2009.

[21] “TPM 2.0: A Brief Introduction,” 2015, last access on 2016-11-01. [Online]. Available: http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-A-Brief-Introduction.pdf

[22] J. Greene, “Intel® Trusted Execution Technology: Hardware-based Technology for Enhancing Server Platform Security,” 2012, Last accessed on 2017-01-30. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf

[23] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, “Innovative technology for CPU based attestation and sealing,” in *HASP - Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, vol. 13, 2013. DOI: 10.1.1.405.8266

[24] “Intel® Software Guard Extensions Remote Attestation End-to-End Example | Intel® Developer Zone,” Last accessed on 2017-05-18. [Online]. Available: https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-example

[25] “Intel® Software Guard Extensions SDK | Intel® Software,” Last accessed on 2016-11-01. [Online]. Available: https://software.intel.com/en-us/sgx-sdk

[26] “Intel® Software Guard Extensions for Linux* OS | 01.org,” Last accessed on 2016-11-01. [Online]. Available: https://01.org/intel-software-guard-extensions/

[27] “Integrity Measurement Architecture (IMA),” Last accessed on 2016-11-14. [Online]. Available: https://sourceforge.net/p/linux-ima/wiki/Home

[28] "SELinux Project Wiki," Last accessed on 2016-11-29. [Online]. Available: https://selinuxproject.org/page/Main_Page

[29] "The SMACK Project: Description from the Linux source tree," Last accessed on 2016-11-29. [Online]. Available: http://schaufler-ca.com/description_from_the_linux_source_tree

[30] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016. DOI: 10.1109/COMST.2015.2453114

[31] N. Paladi and C. Gehrmann, "Towards Secure Multi-tenant Virtualized Networks," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, Aug 2015. DOI: 10.1109/Trustcom.2015.502 pp. 1180–1185.

[32] K. Thimmaraju, B. Shastry, T. Fiebig, F. Hetzelt, J.-P. Seifert, A. Feldmann, and S. Schmid, "Reigns to the Cloud: Compromising Cloud Systems via the Data Plane," *arXiv preprint arXiv:1610.08717*, 2016, Last accessed on 2017-01-29. [Online]. Available: https://arxiv.org/pdf/1610.08717v1.pdf

[33] J. Aumasson and L. Merino, "SGX Secure Enclaves in Practice–Security and Crypto Review," *Black Hat*, 2016.

[34] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions." in *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. New York, NY, USA: ACM, 2013. DOI: 10.1145/2487726.2488370. ISBN 978-1-4503-2118-1 pp. 11:1–11:1, Last accessed on 2017-01-10. [Online]. Available: http://doi.acm.org/10.1145/2487726.2488370

[35] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: trustworthy data analytics in the cloud using SGX," in *2015 IEEE Symposium on Security and Privacy*. IEEE, May 2015. DOI: 10.1109/SP.2015.10. ISSN 1081-6011 pp. 38–54.

[36] N. Paladi and C. Gehrmann, "TruSDN: Bootstrapping Trust in Cloud Network Infrastructure," in *12th EAI International Conference on Security and Privacy in Communication Networks*, 2016, Last accessed on 2017-11-01. [Online]. Available: https://arxiv.org/abs/1702.04143

[37] P. Jain, S. Desai, S. Kim, M.-W. Shih, J. Lee, C. Choi, Y. Shin, T. Kim, B. B. Kang, and D. Han, "OpenSGX: An Open Platform for SGX Research," in

Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, 2016. DOI: 10.14722/ndss.2016.23011 [Online].

[38] “SecureCloud | Specification and implementation of the micro-service framework and API D3.1,” Last accessed on 2017-07-18. [Online]. Available: https://www.securecloudproject.eu/wp-content/uploads/D3.1-final.pdf

[39] P.-L. Aublin, F. Kelbert, D. O’Keeffe, D. Muthukumaran, C. Priebe, J. Lind, R. Krahn, C. Fetzer, D. Eyers, and P. Pietzuch, “Poster: LibSEAL: Detecting Service Integrity Violations Using Trusted Execution,” in *Poster at the Twelfth European Conference on Computer Systems (EuroSys)*. ACM, 2017, Available: https://eurosys2017.github.io/assets/data/posters/poster14-Aublin.pdf, Last accessed on 2017-07-18.

[40] P.-L. Aublin, F. Kelbert, D. O’Keeffe, D. Muthukumaran, C. Priebe, J. Lind, R. Krahn, C. Fetzer *et al.*, “TaLoS: Secure and Transparent TLS Termination inside SGX Enclaves,” Imperial College London, Tech. Rep. 2017/5, Mar. 2017, Available: https://www.doc.ic.ac.uk/research/technicalreports/2017/#5, Last accessed on 2017-07-18.

[41] “TLS for SGX: a port of mbedtls,” Last accessed on 2017-05-18. [Online]. Available: https://github.com/bl4ck5un/mbedtls-SGX

[42] “Intel® SGX SSL,” Last accessed on 2017-07-20. [Online]. Available: https://github.com/01org/intel-sgx-ssl

[43] “wolfSSL with Intel SGX on Linux,” Last accessed on 2017-07-20. [Online]. Available: https://www.wolfssl.com/wolfSSL/Blog/Entries/2017/6/14_wolfSSL_with_Intel_SGX_on_Linux.html

[44] “Scenario: Classic with Open vSwitch | OpenStack Networking Guide,” Last accessed on 2016-11-01. [Online]. Available: http://docs.openstack.org/mitaka/networking-guide/scenario-classic-ovs.html

[45] Y. Xu, W. Cui, and M. Peinado, “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” in *2015 IEEE Symposium on Security and Privacy*. IEEE, May 2015. DOI: 10.1109/SP.2015.45. ISSN 1081-6011 pp. 640–656.

[46] “The STRIDE Threat Model | Commerce Server 2002,” 2005, Last accessed on 2016-11-01. [Online]. Available: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx

[47] A. Håkansson, “Portal of research methods and methodologies for research projects and degree projects,” in *Proceedings of the International Conference on Frontiers in Education : Computer Science and Computer Engineering FECS’13*. CSREA Press U.S.A, 2013. ISBN 1-60132-243-7 pp. 67–73. [Online]. Available: http://kth.diva-portal.org/smash/get/diva2:677684/FULLTEXT02.pdf

[48] “SGX VIRTUALIZATION | 01.org,” Last accessed on 2017-05-17. [Online]. Available: https://01.org/intel-software-guard-extensions/sgx-virtualization

[49] “Best practices for KVM | IBM Knowledge Center,” Last accessed on 2017-06-08. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/linuxonibm/liaat/liaatbpkickoff.htm

[50] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, “Design and implementation of a consolidated middlebox architecture,” in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 24–24, Last accessed on 2017-10-21. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228331

[51] “Setup TLS Connection - Ryu 4.9 Documentation,” Last accessed on 2016-12-22. [Online]. Available: https://ryu.readthedocs.io/en/latest/tls.html

[52] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber, “Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers,” in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, Sept 2016. DOI: 10.1109/NETWKS.2016.7751150 pp. 37–44.

[53] “Introduction Examples for Ryu: SmartSDN Controller (NTT Comware),” Last accessed on 2017-01-30. [Online]. Available: https://osrg.github.io/ryu-book/en/html/introduction_example.html#smartsdn-controller-ntt-comware

[54] “qperf (1) - Linux man page ,” Last accessed on 2017-05-17. [Online]. Available: https://linux.die.net/man/1/qperf

[55] R. Olsson, “Pktgen the linux packet generator,” in *Proceedings of the Linux Symposium, Ottawa, Canada*, vol. 2, 2005, pp. 11–24, Last accessed on 2017-09-23. [Online]. Available: https://ols.fedoraproject.org/OLS/Reprints-2005/LinuxSymposium2005_V2.pdf#page=19

[56] D. Turull, “Open source traffic analyzer,” *Master’s Thesis, KTH Information and Communication Technology*, 2010, Last accessed on 2017-09-23. [Online]. Available: https://people.kth.se/~danieltt/pktgen/docs/DanielTurull-thesis.pdf

[57] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, “Moongen: A scriptable high-speed packet generator,” in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC ’15. New York, NY, USA: ACM, 2015. DOI: 10.1145/2815675.2815692. ISBN 978-1-4503-3848-6 pp. 275–287, Last accessed on 2017-10-01. [Online]. Available: http://doi.acm.org/10.1145/2815675.2815692

[58] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, “The Click modular router,” *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000. DOI: 10.1145/354871.354874 Last accessed on 2017-10-21. [Online]. Available: http://doi.acm.org/10.1145/354871.354874

[59] “The R Project for Statistical Computing ,” Last accessed on 2017-05-17. [Online]. Available: https://www.r-project.org/

[60] “Intel® Software Guard Extensions SDK for Linux * OS | Developer Reference,” Last accessed on 2017-05-19. [Online]. Available: https://download.01.org/intel-sgx/linux-1.6/docs/Intel_SGX_SDK_Developer_Reference_Linux_1.6_Open_Source.pdf

[61] “mbed TLS |Supported SSL / TLS ciphersuites,” Last accessed on 2017-07-23. [Online]. Available: https://tls.mbed.org/supported-ssl-ciphersuites

[62] “mbed TLS 2.0 defaults implement best practices,” Last accessed on 2017-07-23. [Online]. Available: https://tls.mbed.org/tech-updates/blog/mbedtls-2.0-defaults-best-practices

[63] “OpenFlow Protocol API Reference, RYU,” Last accessed on 2017-05-23. [Online]. Available: https://ryu.readthedocs.io/en/latest/ofproto_ref.html

[64] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, “Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS),” Internet Request for Comments, vol. RFC 4492 (INTERNET STANDARD), May 2006, Last accessed on 2017-10-01. [Online]. Available: https://rfc-editor.org/rfc/rfc4492.txt

[65] P. Chown, “Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS),” Internet Request for Comments, vol. RFC 3268

(INTERNET STANDARD), Jun. 2002, Last accessed on 2017-10-01. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3268.txt

[66] M. Frigge, D. C. Hoaglin, and B. Iglewicz, "Some Implementations of the Boxplot," *The American Statistician*, vol. 43, no. 1, pp. 50–54, 1989. [Online]. Available: http://www.jstor.org/stable/2685173

[67] "Pointer Handling | Intel® Software Guard Extensions SDK," Last accessed on 2017-06-18. [Online]. Available: https://software.intel.com/en-us/node/708975?language=en

[68] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Request for Comments, vol. RFC 5246 (INTERNET STANDARD), Aug. 2008, Last accessed on 2017-10-01. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5246.txt

[69] "Intel® Server System LR1304SPCFSGX1," Last accessed on 2017-10-25. [Online]. Available: https://ark.intel.com/products/98607/Intel-Server-System-LR1304SPCFSGX1

[70] "First Cloud Provider to offer Skylake — the latest Intel® Xeon® scalable processor," Last accessed on 2017-10-25. [Online]. Available: https://cloud.google.com/intel/

[71] "Google Compute Engine FAQ | About Google Compute Engine," Last accessed on 2017-10-25. [Online]. Available: https://cloud.google.com/compute/docs/faq

[72] Kai Huang, "KVM-SGX Introduction," Last accessed on 2017-10-25. [Online]. Available: https://github.com/intel/kvm-sgx/wiki

[73] Mark Russinovich, CTO, Microsoft Azure, "Introducing Azure confidential computing," Last accessed on 2017-10-25. [Online]. Available: https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/

[74] S. Morgan, "Cyber Crime Costs Projected To Reach $2 Trillion by 2019," Forbes, Published 2016-01-17, Last accessed on 2017-01-30. [Online]. Available: http://www.forbes.com/sites/stevemorgan/2016/01/17/cyber-crime-costs-projected-to-reach-2-trillion-by-2019/#47610a773bb0

TRITA-ICT-EX-2017:195