Evaluation of Using Secure Enclaves in Virtualized Radio Environments

Emil Norberg

Supervisor: Ulf Kargén
Examiner: Prof. Nahid Shahmehri

External supervisors: Dr. Rahul Hiran & Hampus Tjäder
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Emil Norberg
Abstract

Virtual Network Functions (VNFs) are software applications that process network packets in virtualized environments such as clouds. Using VNFs to process network traffic inside a cloud, which could be controlled by a third-party, exposes the secrets that are stored within the VNFs to a significant amount of threats. Trusted Execution Environments (TEEs) are hardware technologies dedicated to protect software from other malicious applications and users. Open Enclave and Asylo are two SDKs that decouple software and hardware and enable developers to build applications that utilize TEEs without creating hardware dependencies. Open Enclave and Asylo are still in an early stage of development, Asylo in particular. The impact of integrating Open Enclave and Asylo to VNFs from a security and performance perspective was addressed by performing a risk assessment and running performance experiments. The identified vulnerabilities in VNFs were mitigated by using available security properties from TEEs. The results show that protecting VNFs with Open Enclave and Asylo mitigate a significant amount of threats. However, the VNFs suffer from a performance penalty when using TEEs, and are still vulnerable to side-channel and Denial-of-Service attacks.
Acknowledgments

I want to thank my supervisors at Ericsson AB, Dr. Rahul Hiran and Hampus Tjäder, and my supervisor and examiner at Linköping University, Ulf Kargén and Prof. Nahid Shahmehri, for providing me with excellent guidance and supervision. I would also like to thank my family and friends for always providing me with the support I need.

Linköping, May 2019
Emil Norberg
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Aim</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Research questions</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Method</td>
<td>4</td>
</tr>
<tr>
<td>2 Background</td>
<td>5</td>
</tr>
<tr>
<td>2.1 European Telecommunication Standards Institute</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Cloud computing</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Lawful Interception</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Software Defined Networks</td>
<td>8</td>
</tr>
<tr>
<td>2.5 Side-channel attacks</td>
<td>9</td>
</tr>
<tr>
<td>2.6 VM escape attack</td>
<td>9</td>
</tr>
<tr>
<td>2.7 Trusted computing base</td>
<td>9</td>
</tr>
<tr>
<td>2.8 Attestation</td>
<td>9</td>
</tr>
<tr>
<td>2.9 Trusted Platform Module</td>
<td>10</td>
</tr>
<tr>
<td>2.10 Linux Integrity Measurement Architecture</td>
<td>11</td>
</tr>
<tr>
<td>2.11 Intel’s Enhanced Privacy Identification</td>
<td>11</td>
</tr>
<tr>
<td>2.12 Intel’s Safe Guard Extension</td>
<td>12</td>
</tr>
<tr>
<td>2.13 Alternatives to SGX</td>
<td>16</td>
</tr>
<tr>
<td>2.14 Open Enclave &amp; Asylo</td>
<td>17</td>
</tr>
<tr>
<td>2.15 gRPC</td>
<td>17</td>
</tr>
<tr>
<td>2.16 Bazel</td>
<td>18</td>
</tr>
<tr>
<td>2.17 Risk assessment</td>
<td>18</td>
</tr>
<tr>
<td>3 Related work</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Security challenges with VNFRs</td>
<td>19</td>
</tr>
<tr>
<td>3.2 Protecting code confidentiality with Asylo</td>
<td>19</td>
</tr>
<tr>
<td>3.3 SGX performance</td>
<td>20</td>
</tr>
<tr>
<td>3.4 VNFR performance with SGX</td>
<td>20</td>
</tr>
<tr>
<td>3.5 Provisioning secrets to VNFRs with SGX</td>
<td>20</td>
</tr>
<tr>
<td>3.6 Protecting against isolation failure with SGX</td>
<td>20</td>
</tr>
<tr>
<td>3.7 vTPM issues addressed with SGX</td>
<td>20</td>
</tr>
</tbody>
</table>
# List of Figures

2.1 NFV architecture designed by ETSI .................................. 6  
2.2 The x86 architecture’s privilege levels. .......................... 10  

4.1 The X.509 certificate and remote attestation report are sent to the CA .................. 23  
4.2 Overview of the involved attackers, actors and components. Communication between the VNFs is secured with TLS 1.3 ................................................. 25  
4.3 All vulnerabilities available for an attacker were used for finding attacks against the assets. Note that vulnerabilities can be shared between many attackers (see VU06) .................................................. 27  
4.4 Illustration of how the experiments were performed ............................................. 32  

5.1 The VNF’s decryption key is accompanied with the signed certificate .................... 38  
5.2 Screenshot of running Open Enclave’s helloworld example ................................ 40  
5.3 Screenshot of running Asylo’s quickstart example ............................................. 40  
5.4 Results of performance experiments with a simulated SGX .................................. 41  
5.5 Scatter plot of remote Asylo and Open Enclave samples with a simulated SGX ... 42  
5.6 Scatter plot of local Asylo and Open Enclave samples with a simulated SGX ....... 42  
5.7 Results of performance experiments with a hardware SGX in debug mode .......... 43  
5.8 Scatter plot of remote Asylo and Open Enclave samples with a hardware SGX in debug mode ................................................................. 43  
5.9 Scatter plot of local Asylo and Open Enclave samples with a hardware SGX in debug mode ................................................................. 44  

6.1 Scatter plot of 5000 Asylo measurements with a simulated SGX ....................... 47  
6.2 Scatter plot of 5000 Open Enclave measurements with a simulated SGX .......... 48  
6.3 Latency with a simulated SGX .............................................. 48  
6.4 Latency with a hardware SGX in debug mode ............................................... 49
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Snapshot of Open Enclave's and Asylo's github repositories.</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>The top five TEEs (if only supported security properties are considered) compared by Maene et al. [2]</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Assets targeted by the attackers.</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>List of experiments for disclosing the relationship between packet sizes and RTTs without using TEE functionality</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>List of experiments for disclosing the relationship between packet sizes and RTTs with a simulated SGX</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>List of experiments for disclosing the relationship between packet sizes and RTTs with a hardware SGX in debug mode</td>
<td>29</td>
</tr>
<tr>
<td>4.5</td>
<td>Summary of the machine that was used for the experiments.</td>
<td>33</td>
</tr>
<tr>
<td>5.1</td>
<td>Identified VNF vulnerabilities.</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>Threats posed by the external attacker.</td>
<td>35</td>
</tr>
<tr>
<td>5.3</td>
<td>Threats posed by the internal attacker.</td>
<td>36</td>
</tr>
<tr>
<td>5.4</td>
<td>Threats posed by the insider attacker.</td>
<td>36</td>
</tr>
<tr>
<td>5.5</td>
<td>Vulnerabilities mitigated by the isolation protection.</td>
<td>37</td>
</tr>
<tr>
<td>5.6</td>
<td>Vulnerabilities mitigated by using the proposed code confidentiality protection.</td>
<td>38</td>
</tr>
<tr>
<td>5.7</td>
<td>Vulnerabilities mitigated by using the proposed local certificate distribution protocol.</td>
<td>38</td>
</tr>
<tr>
<td>5.8</td>
<td>Vulnerabilities mitigated by the regulatory compliance protection.</td>
<td>39</td>
</tr>
<tr>
<td>5.9</td>
<td>Supported functionality in Open Enclave and Asylo.</td>
<td>39</td>
</tr>
<tr>
<td>6.1</td>
<td>Summary of mitigated threats and vulnerabilities if SGX is used.</td>
<td>46</td>
</tr>
</tbody>
</table>
1 Introduction

The number of connected devices is consistently increasing and challenge the idea of how modern Radio Access Networks (RANs) are structured. By introducing smaller cells, base stations can operate on higher frequencies and provide higher throughput to connected devices. However, smaller cells imply a higher fluctuation of connected mobile devices, which means that during some periods, the base station will consume power while no users are connected [3]. More importantly, it causes poor utilization of available resources.

If a collection of base stations instead shared the same computing center, the problem with wasted computing resources could be much more efficiently handled by allocating resources to base stations that need more attention. This is realized in Cloud-RAN (C-RAN) where base stations are connected to a cloud environment through, e.g., fiber links [3]. To optimize this arrangement even further, other promising technologies such as Virtual Network Functions (VNFs) can be used to make deployment of new features easier and increase the utilization of available resources (used in Virtual-RAN, also referred to as V-RAN).

1.1 Motivation

VNFs enable mobile network operators (referred to as operators from now) to distribute the workload generated by network traffic over multiple VNFs that has been allocated in a cloud. If there is a significant increase in network traffic, operators can allocate more VNFs to sustain the workload. However, not all operators may possess the financial resources to host their own cloud environment. Allocating the VNFs in a shared cloud is, therefore, a more suitable solution for the operators.

The National Institute of Standards and Technology (NIST) has divided the cloud into four different deployment models [4]:

- **Public cloud**: Anyone can be the Cloud Service Provider (CSP) and everyone has access to the cloud service.

- **Private cloud**: Only one organization (e.g., government or company) has access to the cloud service.

- **Community cloud**: A group of organizations shares the cloud service.
• **Hybrid cloud:** The cloud environment is comprised of two or more of the deployment models mentioned above.

There are many threats to consider when outsourcing computations to a public, community, or hybrid cloud. For instance, the underlying infrastructure could be compromised by an attacker [5], or an employee working at the cloud provider could steal confidential data [6]. According to NIST, the security challenges are particularly difficult for public clouds because a third-party controls the underlying infrastructure [7]. Coppolino et al. [5] divide the current threats towards cloud applications into three different attack vectors:

• **External attacks:** An attacker can target tenants’ network access to hijack their connection and take control over their account.

• **Internal attacks:** An attacker can deploy a malicious application on the CSP and try to gain access to tenant accounts by attacking vulnerabilities in the underlying infrastructure (e.g., the hypervisor).

• **Insider attacks:** Employees working at the CSP can tamper with the hardware to steal confidential data from tenants.

The attack vectors mentioned above are for cloud applications in general. In the paper by Lal et al. [8], threats towards VNFs are presented and divided into six different categories (see list below). Note that every threat can be mapped to one of the three aforementioned attack vectors.

• **Isolation failure:** An attacker compromises the hypervisor by attacking a VNF that is deployed on top of it (VM escape attack). If the attacker successfully gains control over the hypervisor, all other VNFs running on the same hypervisor can be considered to be compromised as well.

• **Network topology validation and implementation failure:** Given that an attacker successfully compromises, e.g., a virtual firewall, the attacker could alter the firewall settings such that an attack could be performed on another target that was not reachable before.

• **Regulatory compliance failure:** An attacker steals the VNF’s code and deploys it in a country where the VNF violates regulatory compliance and, therefore, successfully puts the VNF’s owner in a difficult situation.

• **Denial of Service protection failure:** An attacker can deploy multiple malicious VNFs on the same hypervisor as a legitimate VNF and use all the available resources (CPUs, network connections, ...) to perform a Denial of Service (DoS) attack.

• **Security logs troubleshooting failure:** An attacker can deploy multiple malicious VNFs on the same hypervisor as a legitimate VNF with the purpose of overflowing the hypervisor log until specific entries in the log have been deleted. Such an attack could be useful when the attacker has successfully performed an attack on a legitimate VNF and wants to hide its tracks by removing all the evidence.

• **Insider attacks:** An attacker that works as an employee at the cloud center might perform an attack on the hardware (e.g., probing and fault-injection) to extract confidential data.

Using a Trusted Computing Platform (TCP) for providing security to tenants in cloud environments is proposed as a solution in many research papers ([9], [10] and [11]). The Trusted Platform Module (TPM) is an example of a TCP that provide users with several useful
1.2. Aim

Tools, e.g., secure booting and storing keys [11]. Unfortunately, TPMs cannot be shared by Virtual Machines (VMs) [12] without using a virtual TPM (vTPM), which is not as secure as using a normal TPM [13].

A Trusted Execution Environment (TEE) can be viewed as an enhanced TPM with additional features for executing applications isolated from the OS kernel [14]. Many hardware manufacturers have already implemented support for TEEs, e.g., Intel’s Safe Guard Extension (SGX) [15] and ARM TrustZone [16]. Compared to only using software protection as a countermeasure to VNF threats, TEEs provide a higher level of security because the protection stretches down to the hardware [17].

Creating applications that utilize hardware support for one type of processor is terrible for portability. Therefore, a more suitable solution would be to add a layer between the software and hardware such that applications can be developed independently from the hardware but still use TEE functionality. Open Enclave [18] and Asylo [19] are two Software Development Kits (SDKs) that are aiming for fulfilling this functionality. Unfortunately, both SDKs are in an early phase of development and only supports SGX, for now.

There are published papers regarding how keys can be provisioned securely to VNFs protected by TEEs [20]. However, more research is required on how it can be done without compromising other important aspects that need to be fulfilled if the VNF should be possible to be used in production, such as portability, security, and performance. By using Open Enclave and Asylo, the TEE hardware is abstracted, which solves the portability problem. The impact on security and performance is, however, not addressed, and should be addressed before the SDKs are used in production.

To conclude, there are many benefits of using VNFs for cloud applications but the security issues that follow cannot be neglected. If Asylo or Open Enclave, two SDKs that provide developers with great opportunities for creating enclave applications can be used to address these issues then it will be a great contribution to cloud security.

1.2 Aim

To distinguish the difference between Asylo and Open Enclave, and determine their level of maturity, a deeper analysis is needed. Many aspects could be considered during such an analysis, but some have higher priorities than others. The aspects that are considered to have the highest priority, in this thesis, are the ones that need to be fulfilled for being able to use the SDKs with already existing VNFs in production. More specifically, this thesis is focusing on security and performance impacts of:

- Running a VNF protected by Open Enclave or Asylo in a public, community, or hybrid cloud.
- Provisioning credentials, such as keys or a signed certificate, to VNFs protected by Open Enclave or Asylo in a public, community, or hybrid cloud.

1.3 Research questions

Security is essential, particularly for telecommunication purposes, where all data exchanged between users should be confidential. TLS 1.3 is a popular protocol for establishing secure communication channels between two endpoints and is also used by other related work for securing the communication between VNFs [20]. If the X.509 certificates (used in TLS for establishing a secure communication channel) cannot be distributed to the VNFs without disclosing the certificate’s private key to an attacker, then the confidentiality of exchanged information is compromised. It is therefore important to address the impact on security if Open Enclave and Asylo are used for protecting VNFs.
Q1 What is the impact of integrating Open Enclave and Asylo to VNFs from a security perspective?

Telecommunication systems are known to have hard real-time constraints for being able to provide a service with high throughput. Therefore, it would be interesting to measure the performance overhead of running a VNF protected by Open Enclave and Asylo compared to when no SDK is used.

Q2 What is the impact of integrating Open Enclave and Asylo to VNFs from a performance perspective?

1.4 Method

Research question Q1 was addressed by first identifying the security properties necessary for protecting a VNF during runtime or provisioning of credentials. The required security properties for protecting VNFs were determined by performing a risk assessment. When the security properties had been identified, a literature study was conducted on the available examples and documentation about the SDKs to verify that the necessary security properties are supported. The number of supported security properties in the SDKs were used as a metric for determining the impact on security if Open Enclave or Asylo is used.

Research question Q2 was addressed by running the performance experiments listed below. The implementation used for the experiments signed packets with a SHA-256 hash function and a secret key that was located within the enclave.

- No TEE functionality (baseline)
- Simulated SGX with local measurements
- Simulated SGX with remote measurements (TEE functionality is located in a separate container)
- Hardware SGX in debug mode with local measurements
- Hardware SGX in debug mode with remote measurements
This chapter contains the necessary background information for understanding the rest of this report. It starts with general information about cloud environments, VNFs, and the European Telecommunication Standards Institute (ETSI). The security-related topics are then presented with background information about Open Enclave, Asylo, and the frameworks used by the SDKs. Background information that is needed for understanding the methodology chapter is included at the end (see Section 2.17).

2.1 European Telecommunication Standards Institute

ETSI [21] is an European standards organization focused on Information and Communication Technology (ICT). ETSI has formulated a Network Function Virtualization (NFV) architecture (see Figure 2.1) for the Telecommunication Cloud (Telco Cloud), which is comprised of the following four domains [22]:

- Management and Operations (see Section 2.1)
- NFV Infrastructure (see Section 2.1)
- VNF layers (see Section 2.1)
- Operating/Business Systems Support (OSS/BSS); outside the scope of this thesis, and is therefore not covered in this background chapter.

Element management

The Element Management (EM) domain enable the OSS/BSS to perform management operations on VNFs, such as security configurations and collect performance statistics. [23]

Virtual Network Functions

VNFs are virtualized Network Functions (NFs) implemented as software applications that do not require the physical equipment to be installed on the platform where the software is executed. The Dynamic Host Configuration Protocol (DHCP), Network Address Translation
(NAT), Packet Processing (PP) and Intrusion Detection Systems (IDSs) are examples of NFs that can be used as VNFs.\cite{24}

Instead of deploying NFs physically in Customer Premise Equipment (CPE), VNFs enable, e.g., an Internet Service Provider (ISP) to centralize all physical network functions from the CPEs into one shared data center. The ISPs can then easily update the NFs without the need to send out technicians to each customer, which is not only expensive but also inefficient.\cite{24}

**Lifecycle**

When a VNF is running inside a VM, certain Life Cycle Management (LCM) operations need to be supported to avoid unexpected interruption of service.\cite{25} Note that the following operations are outside the scope of this thesis but could be interesting to investigate further from a security perspective in future work:

- **Migration**: VNFs are stopped and relocated to another hypervisor (could be in another cloud service provider).\cite{25}
- **Suspend**: The VNF’s execution is halted, and a snapshot is stored in memory.\cite{25}
- **Resume**: The VNF’s snapshot is loaded from memory and resumed.\cite{25}

**VNF component**

To simplify the scaling of resources within a VNF, the VNFs functionality can be divided into smaller components, also referred to as VNF Components (VNFCs). A general VNF can be a collection of multiple VNFCs. According to the specifications provided by ETSI, VNFCs may be deployed in a public cloud that is owned by a third-party, which increases the number of
2.2. Cloud computing

There are two participants in cloud computing: a CSP and a tenant. CSPs enable tenants to quickly allocate and release computing resources in computer centers through network access. According to NIST [4], services offered by CSPs can be one of the following three service models:

Software as a Service

Tenants are only authorized to interact with applications hosted on the CSP. A web application is an example of a service usually offered to tenants through Software as a Service (SaaS). Tenants are not authorized to change the application, which includes the operating system, storage mediums, network functions, and underlying infrastructure (e.g., hypervisor).
2.3. Lawful Interception

**Platform as a Service**

Tenants are authorized to deploy software applications on the CSP and use libraries supported by the platform. Tenants do not have access to change the operating system, storage mediums, network functions, or underlying infrastructure.

**Infrastructure as a Service**

Tenants are authorized to alter almost everything above the virtualization layer, which includes the operating system, storage mediums, and network functions, but not the underlying infrastructure.

**Telecommunication cloud**

The following service models are defined by ETSI and extend the service models defined by NIST:

**NFVI as a Service**

Operators may not possess the necessary resources to offer global customers VNFs because of the large costs such an infrastructure would require. CSPs, on the other hand, may already own the necessary infrastructure and can, therefore, help operators by offering them NFVI as a Service (NFVIaaS).

NFVIaaS could potentially be deployed with any NIST deployment model, but according to ETSI, the public cloud is not expected to be used because of the potential effects on performance and throughput.[22]

**VNF as a Service**

VNF as a Service (VNFaaS) is an extension of SaaS where the hosted applications are VNFs. By using VNFaaS, organizations can acquire the network functionality needed (e.g., Access Routers and Firewalls) without maintaining or hosting the NFs.[27]

2.3 Lawful Interception

CSPs that offer NFVIaaS needs to support Lawful Interception (LI) functionality such that Lawful Enforcement Agencies (LEAs) can intercept the traffic of a VNF. ETSI’s proposed solution is to allow LEAs to allocate LI-VNFs on the NFVI with the necessary privileges to interrogate other VNFs if the LEA can provide a warrant. LI-VNFs should have access to intercept the communication during a limited amount of time that is specified in the warrant, and the target should always remain unaware of the fact that its traffic is intercepted by an LEA.[28]

2.4 Software Defined Networks

The purpose of Software Defined Networking (SDN) is to separate the data plane from the control plane and to simplify network management such that network nodes, switches, and other network components can be easily configured from one centralized controller. Network administrators do not need to configure network nodes individually, which is not only time consuming but also requires a certain set of skills.
2.5 Side-channel attacks

Side-channel attacks exploit information leakages in hardware for being able to compromise secrets [30]. In a cloud environment, tenants may share the same hardware resources, which makes it difficult to isolate tenants such that there are no information leakages between tenants, i.e., side-channel vulnerabilities. There exist a lot of research that illustrates the effectiveness of side-channel attacks:

- Zhang et al. [31] showed how a tenant’s private key could be revealed by using a cache-based side-channel attack. The cache-based side-channel attack typically exploits information leakage in the CPU’s cache and can be performed from a malicious application that shares the same machine as the victim.

- Messerges et al. [32] showed how differential power analysis could be used for revealing keys located inside a smartcard. The differential power analysis can be used as a hardware-based side-channel attack and is performed by monitoring a device’s power consumption with the purpose to extract secrets.

2.6 VM escape attack

By utilizing CPU virtualization features and hardware Virtual Machine Extensions (VMXs), tenants can use the same hardware by mounting VMs on, e.g., a hypervisor. The x86 architecture’s privilege levels are visualized in Figure 2.2 and is used in many existing IaaS deployments. As illustrated in Figure 2.2, VMX root Ring 0 has the highest privileges and VMX non-root Ring 3 has the lowest privileges.[1]

Applications can read and modify both data and software that is running with lower privileges.[1]. Attackers with high privileges can, therefore, have access to other users’ resources. If the attacker does not have the necessary privileges to compromise other users’ data, the attacker can try to exploit vulnerabilities in software that is running with higher privileges such that the attacker’s privileges escalate to the same privileges as the compromised software. Similar attacks exist in cloud services, where the attacker provisions a VM and escalate its privileges by exploiting a vulnerability in the hypervisor. This type of attack is referred to as a VM escape attack, and threatens the isolation of tenants in cloud environments.[8]

2.7 Trusted computing base

The Trusted Computing Base (TCB) was defined by John Rushby in a conference paper from 1984 [33]. Rushby’s definition of the TCB can be summarized as following: a TCB utilize hardware and software resources to protect the system and itself from being compromised by an adversary.

2.8 Attestation

According to Coker et al. [34], attestation can be used for verifying properties of a remote system (e.g., a TCB). It can be particularly useful for use cases where the verifier is about to
send sensitive information to a remote system but wants to authenticate that the remote system can be trusted before secrets are sent. Attestation is implemented as a security measure for verifying a remote system in many popular systems, such as TPMs (see Section 2.9) and Intel’s SGX (see Section 2.12).

2.9 Trusted Platform Module

TPMs are hardware modules designed according to the Trusted Computing Group’s (TCG) specifications and can be installed on a motherboard for securing Trusted Building Blocks (TBBs), i.e., components that are expected to provide a Root-of-Trust (RoT). TPMs are not TCBs, but they can be used for determining if a TCB has been compromised. The TPM is used for establishing the following three RoTs:

- **Root of Trust for Measurement (RTM)**: Measurements in the context of software integrity, are blocks of code digested with a hash function, e.g., SHA-256. TPMs continuously measures code and stores the measurements in Platform Configuration Registers (PCRs) such that after the machine has finished booting, all the software running on the machine is represented by the PCRs. New measurements always extend the current measurements by applying the following formula:

  \[ PCR[N + 1] = \text{hash}(PCR[N] || \text{Measurement}) \]

  The RoT for Measurement (RTM) is a TBB responsible for generating accurate measurements of the software running on the attached machine. The measurement of a machine can be divided into the following three parts:

  - **Core RoT for Measurement (CRTM)**: Measurement of the initial block of code that is executed on the machine
  - **Static RoT for Measurement (SRTM)**: Measurement of the Basic Input/Output System (BIOS)
  - **Dynamic RoT for Measurement (DRTM)**: Measurement of e.g., the operating system and drivers

- **Root of Trust for Storage (RTS)**: The RoT for Storage (RTS) is a TBB responsible for storing data securely. The TPM is equipped with hardware protection against tampering and can, therefore, be used as an RTS. TPMs also
2.10 Linux Integrity Measurement Architecture

supports other features such as storing data that is only available if the PCRs contain a certain set of measurements (also known as sealing). [35]

Root of Trust for Reporting

Every TPM is equipped with an Endorsement Key (EK). The EK corresponds to an Endorsement Certificate (EC), which can be used for proving the authenticity of the TPM. A remote party can authenticate a TPM by sending a string encrypted with the EC’s public key as a challenge. If the TPM can produce the plaintext, then the TPM has successfully proven its ownership of the EK and EC. [35]

The EK can be used for enrolling additional keys, e.g., keys for signing attestation reports. If an attestation key is enrolled, it can be used for signing attestation reports that contain information about the TPM, such as the TPM’s PCRs.

The RoT for Reporting (RTR) enables users to retrieve data that is stored inside the RTS securely. Signing attestation reports that contain authentic data is one example of the RTR’s responsibilities. [35]

Using TPMs in virtualized environments

TPMs can be used to continuously measure software before it is executed, and thereby generate a chain of trust that starts with the first block of code executed on the machine. Applications that are mounted on the host have the opportunity to seal secrets such that they can only be decrypted if the exact same software was booted in the exact same order. In a virtualized environment, many virtual machines are expected to be running in parallel, and since the VMs could have been measured in different orders every time they were started, there is a low probability that sealed secrets by the VMs can ever be decrypted again. If each VM occupied the hypervisor until its execution was finished, this problem would have been resolved. However, if each VM occupied the hypervisor until it finished, then the environment may not be considered to be virtualized anymore. Therefore, the TPM was not considered to be suitable for securing VNFs in environments that are virtualized, such as cloud environments.

2.10 Linux Integrity Measurement Architecture

The Linux Integrity Measurement Architecture (IMA) can be used for detecting intrusions after the machine has finished booting. Files that should be measured by the IMA can be configured from a policy file. If a TPM is installed on the motherboard, the measurements can be stored inside the PCRs. If no TPM is installed, then the measurements are not protected from being tampered with by an attacker. [37]

2.11 Intel’s Enhanced Privacy Identification

The Enhanced Privacy IDentification (EPID) is a technology invented by Intel, dedicated to making processors that subscribe on premium content anonymous toward the premium content provider. It also addresses some of the existing issues with Public Key Infrastructures (PKIs) such as not being able to revoke certificates if the private key has been compromised. [38]

Anonymity

Platforms can remain anonymous to the rest of the EPID scheme thanks to the use of a group signature scheme. In a group signature scheme, many private keys can share the same public key, which makes it impossible for a verifier to decide which platform that signed a message because any member of the EPID group can have signed the message. [38]
Roles

There are mainly three different roles involved with the EPID scheme (EPID authority, Verifier and Platform) but this section will also include one additional role that plays an important part of the EPID scheme: the Online Certificate Status Protocol (OCSP) servers:

Online Certificate Status Protocol servers

The OCSP servers are responsible for providing verifiers and platforms with new lists of revoked private keys, signatures and group IDs. The lists are signed by the EPID authority which ensures that the OCSP servers cannot forge new revocation lists. Each OCSP server is provisioned a signed X.509 certificate by the EPID authority for being able to prove that the revocation lists originate from a trusted party.[38]

When a verifier is about to establish a secure communication channel with a platform, the platform can include an OCSP challenge to ensure that the received lists of revoked certificates are fresh (another word for immune to replay attacks). The verifier then sends the challenge to an OCSP server and receives a new set of revoked certificates accompanied with a response for the challenge.[38]

EPID Authority

The EPID authority is responsible for many parts of the EPID scheme:

- **Generating and storing key pairs:** Each platform is a member of a group that corresponds to a group public key (group ID) and group master private key. The group master private key is used for generating private keys for every platform that is a member of the group. After the private key has been stored inside the platform, the EPID authority destroys its copy of the private key. The master private key is stored along with the group ID, because otherwise, the EPID authority will not be able to produce new private keys if, e.g., an entire group ID is revoked.[38]

- **Revoking private keys, signatures or group IDs:** The EPID authority decides which private keys, signatures and group IDs that should be revoked. If, e.g., a private key is revoked, then the OCSP (see Section 2.11) servers receive a new list of revoked private keys, signatures and group IDs from the EPID authority.[38]

Verifier

The verifier can be a service provider that broadcasts premium services for a group of platforms. The verifier has an X.509 certificate signed by the EPID authority that is used for authenticating itself towards the platforms.[38]

Platform

The platform is a device that subscribes on a premium service offered by the verifier. The platform proves that it has permission to access the service by using the private key that was provisioned to the platform by the EPID authority.[38]

2.12 Intel’s Safe Guard Extension

Intel’s SGX enables applications to execute with confidentiality and integrity, even if the kernel or other privileged applications running on the machine are controlled by an intruder [1]. All computations that should be protected are executed inside a protected space that is usually referred to as an enclave or TEE. An enclave can be summarized as a safe space inside the processor where the code can execute in isolation.[1]
Measurement

The measurement process begins when the enclave starts booting. A measurement is represented by the enclave’s previous states by continuously digesting data that is related to the enclave’s current state, e.g., location of memory, memory content, and related security flags. The measurement can only be updated during the booting process of the enclave, and the commands used for updating the measurement is disabled when the enclave has been launched. When the enclave has been launched, the resulting measurement represents the enclave’s identity.\[1\]

Attestation key

Each SGX platform is provisioned an Intel EPID member key, also referred to as the attestation key, which is used for signing remote attestation reports that can be verified by the Intel Attestation Service (IAS). The attestation key is received by the Provisioning enclave (see Section 2.12) from the Intel Provisioning Service (IPS) if the Provisioning enclave can prove that it is a Provisioning enclave. This is done by sending the Provisioning secret that is burnt into the e-fuses during manufacturing. If the Provisioning secret is a legitimate token generated by Intel, the Intel Provisioning Service produces an attestation key and sends it encrypted to the Provisioning Enclave. The Provisioning enclave then encrypts the received attestation key using a shared key between the Provisioning enclave and Attestation enclave and sends it to the Attestation enclave.\[1\]

Attestation

SGX supports both remote and local attestation \[1\]. An attestation report contains the following fields (among others):

- **MRENCLAVE**: The enclave’s measurement.\[1\]
- **MRSIGNER**: A SHA-256 digest of the enclave signer’s modulus operator of the public key (RSA is used).\[1\]
- **ISVPRODID**: Product identification number. ISV stands for Independent Software Vendor.\[1\]
- **ISVSVN**: The security version number of the enclave’s software. Is controlled by the enclave’s owner and should be incremented when vulnerabilities are patched.\[39\]
- **CPUSVN**: Security version number related to the SGX hardware.\[39\]
- **REPORTDATA**: Contains user supplied data and is included with the integrity protected section of the attestation report.\[1\]

Remote attestation

The remote attestation protocol is initiated by sending a challenge to the target enclave. The target enclave then produces a measurement report that is passed to the Attestation enclave (see Section 2.12). SGX supports two different methods for signing and verifying attestation reports:

- Each SGX is part of an EPID member group, which means that the platform will remain anonymous during authentication. The public key used for verifying signed measurement reports is the EPID public key corresponding to the group that the platform is a member of. The attestation key used for signing measurement reports is an EPID private key. When the remote party receives the signed measurement report, also referred to as a quote, the signature is verified by using the Intel Attestation Service (IAS).\[1\]
• The SGX can be equipped with Intel Data Center Attestation Primitives (DCAP), which enable users to sign remote attestation reports that can be verified without the IAS. [39]

Local attestation
Local attestation is used between enclaves on the same SGX platform to verify each other’s authenticity. The protocol is similar to the remote attestation protocol, but without the step where the Attestation enclave (see Section 2.12) signs the measurement report. The Attestation enclave is not needed because the two enclaves can use a shared key unique for the platform to verify each other’s authenticity with a Message Authentication Code (MAC). [1]

Built-in enclaves
The following enclaves are running on all SGX platforms [1]:

• **Attestation enclave**: Dedicated for signing remote attestation reports with the Intel EPIID member key, i.e., the report needs to be verified with the Intel Attestation Service (IAS).

• **Provisioning enclave**: Receives attestation key from the Intel Provisioning Service.

• **Launch enclave**: Decides whether an enclave is allowed to be executed as a production enclave.

Intel recently added support that allows organizations to verify attestation reports without using the Intel Attestation Service (IAS). By allowing organizations to verify attestation reports without the IAS, Intel does not have to be trusted anymore for verifying remote attestation reports, and verification can be successfully performed without an internet connection. This is enabled by introducing an additional enclave [39]:

• **Provisioning Certification Enclave (PCE)**: The PCE signs attestation certificates for other local quoting enclaves (may be created by any developer) with a Provisioning Certificate Key (PCK). The PCK is linked to a certificate issued by Intel and is acquired by adding DCAP (see Section 2.12). [39]

Sealing
Sealing is a feature in SGX for encrypting information with a key generated by the EGETKEY instruction [15]. Secrets can be sealed in two different ways:

Sealing with enclave identity
When a secret is sealed to an enclave’s identity, the secret is only accessible by enclaves with the same identity as the enclave that sealed the secret. This also means that even if two enclaves are signed by the same authority, they will not be able to share secrets like in the case of sealing with the sealing identity (see next section). [15]

Sealing with sealing identity
If secrets are sealed with the sealing identity, then the secrets can only be decrypted by enclaves with [15]:

• a higher or equal SVN.

• the same MRSIGNER field (signed with the same private key).
Sealing with the sealing identity is useful in cases where the enclave’s software is patched, and the patched enclave should still have access to the previous enclaves’ secrets. The enclaves’ measurement (enclave identity) would not remain the same after a software patch, and secrets would not be accessible by the patched enclave if the secrets were sealed with the sealing identity.\footnote{15}

**Foreshadow attack**

Van Bulck et al. \cite{40} present a side-channel attack named Foreshadow that compromise the SGX confidentiality protection by exploiting a vulnerability that can be triggered from userspace. This means that attackers can remotely deploy malicious enclaves on a CSP and extract secrets stored inside enclaves on the same SGX platform as the attacker. The authors clearly state that even production enclaves, such as the Attestation enclave, is vulnerable to the Foreshadow attack, which means that the key used for signing attestation reports can be compromised by an attacker \cite{40}. If the attestation key is compromised, no remote attestation report that has been produced on the compromised SGX platform can be trusted because the attacker could have signed the report for a malicious enclave.

**SGX modes**

Enclave applications can be executed in the following four modes:

- **Release mode**: The enclave application is executed with memory protection and cannot be debugged \cite{41}. However, this mode requires a commercial license (see Section 2.12) \cite{41}. In this thesis, enclaves that are running in this mode is also referred to as production enclaves.

- **Debug mode**: The enclave application is executed in debug mode, which means that no compiler optimizations are used and the memory can be debugged, which makes it unsafe to use for production purposes.\cite{42}\cite{41}

- **Pre-release mode**: The enclave application is executed in debug mode but with compiler optimizations. The debug symbols are disabled in this mode.\cite{41}

- **Simulation mode**: The enclave application is executed without the SGX hardware by simulating the SGX instructions with libraries from Intel.\cite{43}\cite{41}

**Commercial license**

It is not possible to execute enclaves with memory protection without possessing a commercial license from Intel \cite{1}. As a consequence, Intel can choose what companies or individuals that should be able to use the SGX technology and has given rise to other TEEs, such as Sanctum (see Section 2.13), that do not require a commercial license.

The SGX platform can still be used even if the user does not possess a commercial license by either simulating the SGX \cite{43} or running the enclave applications in debug mode \cite{42}. If the enclave uses a simulated SGX, or the enclave is running in debug mode, the memory can be debugged and secrets can easily be extracted by an attacker. Therefore, for production purposes, a commercial license is required.

**Performance overhead**

According to Intel’s developer guide \cite{44} for version 2.4, there is a significant performance overhead in the following four cases:

- **Enclave creation**: The enclave’s code is measured during boot and causes a high performance penalty for creating an enclave.\cite{44}
2.13 Alternatives to SGX

- **Enclave transitions:** Entering and exiting an enclave contributes with a significant performance penalty compared to a normal system call. Halting an enclave with an interrupt also causes a larger execution time compared with a normal interrupt since the enclave needs to perform additional security operations before switching to the unsafe environment.[44]

- **Excessive cache misses:** SGX provides additional protection during cache checks and therefore causes a larger execution time for cache misses than a normal no-SGX cache miss.[44]

- **Excessive writing of pages:** The protected memory reserved for the enclave in the Dynamic Random Access Memory (DRAM) is referred to as an Enclave Page Cache (EPC) [1]. If the memory used by the enclave is larger than the EPC, paging is performed such that memory that is available in a secondary storage can be utilized to extend the limited memory provided by the EPC. When a page is evicted to a secondary storage it first needs to be encrypted, which causes a performance penalty compared to a regular paging operation.[44]

Note that dynamic memory allocation is not explicitly stated to be a problem from a performance perspective, even though the experiments by Zhao et al. [45] prove that there is a significant performance overhead when dynamic memory allocation is performed (see Section 3.3).

2.13 Alternatives to SGX

This section presents four alternative TEEs with a similar level of security as SGX (see Section 3.9).

**Intel Trusted Execution Technology**

Intel’s Trusted Execution Technology (TXT) can be used for extending the TPM’s PCRs to applications. However, all execution on the machine is interrupted when the TXT code is running, which includes the operating system and interrupts, and remains suspended meanwhile the TXT is running [2]. It has a negative impact on the performance of other tenants and may even suggest that Intel TXT is not suited for cloud applications.

**Sanctum**

Sanctum address some of the current issues with SGX, such as the side-channel vulnerabilities and not being able to run production enclaves without possessing a commercial license [46]. However, Sanctum is not equipped with a Memory Encryption Engine (MEE) and is therefore vulnerable to, e.g., physical attacks [2].

**AEGIS**

If only the security metrics by Maene et al. [2] are considered, AEGIS support the same security properties as SGX. For being able to use the AEGIS Tamper-Evident (TE) environment, three instructions must be supported by the processor: `enter_aegis`, `exit_aegis` and `sign_msg`. The TE is entered by executing the `enter_aegis` instruction. `sign_msg` can be used for signing a message with the CPU which binds the signature to the program and CPU (because a hash of the currently executed program is included with the signature). [47]
SecureBlue++

SecureBlue++ \[48\] is a TEE developed by IBM that enables processors to execute Secure Executables with confidentiality and integrity without the large overhead in performance (compared to SGX). The Secure Executable is a binary encrypted with an Executable key, which is a public key that is hardcoded inside CPUs that support SecureBlue++. Unlike SGX, SecureBlue++ do not require any changes to the source code \[48\]. In the study by Maene et al. \[2\], SecureBlue++ supports the same security properties as SGX, except the capability to attest the software.

2.14 Open Enclave & Asylo

Microsoft and Google have released SDKs for building TEE applications: Open Enclave \[18\] and Asylo \[19\]. The goal with Asylo and Open Enclave is to abstract TEE hardware such that applications can utilize TEE functionality without only being restricted to one type of TEE hardware. However, the only supported hardware, for now, is SGX \[49\][50].

Table 2.1: Snapshot of Open Enclave’s and Asylo’s github repositories.

<table>
<thead>
<tr>
<th>SDK</th>
<th>Contributors</th>
<th>Issues</th>
<th>Commits</th>
<th>Releases</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Enclave</td>
<td>44</td>
<td>122</td>
<td>3896</td>
<td>4</td>
<td>0.5.0</td>
</tr>
<tr>
<td>Asylo</td>
<td>24</td>
<td>5</td>
<td>755</td>
<td>18</td>
<td>0.3.4.2</td>
</tr>
</tbody>
</table>

Table 2.1 is a snapshot of the current number of contributors, issues, commits, and releases on each SDK’s GitHub repository. Open Enclave has a more significant number of contributors (83%), issues and commits, and may suggest that Open Enclave is further in the development process than Asylo.

According to Google \[19\], one of the ambitions with Asylo is to enable applications to run on TEE hardware without making any changes to the application’s code. Asylo fulfilled this promise in v0.3.4 by implementing the necessary functionality to wrap an application and run it inside an SGX enclave.\[51\]

Asylo is using the build language Bazel (see Section 2.16), which may be a problem for developers that wants to integrate Asylo to an existing project that is using some other build tool such as Make. One alternative solution to resolve this issue is to not include Asylo to an existing building process and instead isolate the TEE functionality inside a separate container. The TEE functionality can still be used by communicating with the enclave over gRPC (see Section 2.15).

2.15 gRPC

gRPC is a Remote Procedure Call (RPC) framework that enables communication over the HTTP/2.0 protocol \[52\], and can be used in several programming languages such as C++, C#, Java, Python, Ruby and NodeJS \[53\]. gRPC is one of the incubating projects managed by the Cloud Native Computing Foundation (CNCF) \[54\], and is used by many other organizations such as Netflix and CISCO \[54\].

According to the available documentation on the gRPC webpage \[55\], there are several supported authentication mechanisms. Users can choose to integrate their own authentication system \[55\], which is good for Asylo since the attestation reports can be used as the underlying technology for authentication. gRPC also supports functionality for communication over TLS \[55\].
2.16 Bazel

Bazel is a high-level build language that abstracts the building process of applications. Bazel can be used for building applications in languages such as Java, Python, and C++. It also supports functionality for extending the building tool for additional languages, in case that the desired language is not currently supported.

2.17 Risk assessment

The Special Publication (SP) 800-30 by NIST describes how risk assessments should be conducted. A risk can be explained by the following relation:

\[ \text{Risk} = \text{Probability} \cdot \text{Impact} \]

A risk can be mitigated in two different ways: (i) the probability that the risk is triggered is decreased or (ii) by reducing the impact if the risk is triggered. Risk assessment is a component of the risk management process, which comprises the following four components:

- **Risk Framing**: The environment is described.
- **Risk Assessment**: Threats and vulnerabilities are identified, and the impact of exercised threats are estimated.
- **Risk Response**: Decisions are made on how the identified risks from the risk assessment should be mitigated.
- **Risk Monitoring**: The risk should be monitored after the risk response has been implemented because the environment might change such that a new risk response needs to be implemented.

NIST has defined three different approaches for conducting risk assessments:

- **Threat-oriented**: Threat sources (attackers) and threat-events (series of steps for compromising an asset) are identified and used for finding threats.
- **Asset/impact-oriented**: Assets and impacts are used as a starting point to identify threat events that can be exercised by threat sources.
- **Vulnerability-oriented**: Threats are identified by analyzing the existing vulnerabilities that could be exercised by an attacker.
Related work

This chapter presents work that has already been conducted in fields related to this thesis. The only source that mentions Asylo or Open Enclave is located in Section 3.2.

3.1 Security challenges with VNFs

Lal et al. [59] describe the security challenges with running VNFs in a CSP that is not owned by a trusted party. Protecting the secrets from the CSP (insider attackers) and other VMs on the same machine (internal attackers) is a difficult problem. Another challenge is to establish secure communication channels between VNFs, which can be within the same data center or between two different data centers.

Coppolino et al. [5] present several threats against cloud applications in general. Three different attack vectors are presented: internal, external, and insider attacks, which were also mentioned in the introduction. Lal et al. [8] present several best practices to mitigate VNF threats, such as signing VNF images and using remote attestation for verifying the software that is running on the cloud. None of the mentioned best practices suggested that TEEs could be used as a countermeasure.

3.2 Protecting code confidentiality with Asylo

Lazard et al. [60] present a framework named TEEshift that can be used for protecting code confidentiality without making any changes to the source code. The functions that should be protected is pointed to by ELF symbols in a file that the developer supplies as input to TEEshift. Asylo is used for generating the enclave code. When TEEshift is executed, it encrypts the functions inside the binary according to the input file supplied by the developer.

The binary can only be executed on the remote host if it has access to the decryption key because CPUs cannot execute encrypted binaries. To preserve the confidentiality of protected functions, the remote host first loads the encrypted functions into the TEE, which is attested by the application vendor before the decryption key is sent. The protected functions can then be executed along with the remaining application with both confidentiality and integrity, without revealing the code of the functions that were deployed encrypted. [60]
3.3 SGX performance

Zhao et al. [45] performed experiments on SGX to measure its performance for OCALLS (function calls from the enclave to the untrusted memory), ECALLS (function calls from the untrusted memory to the enclave) and allocation of memory inside the enclave. The authors concluded that the number of cycles per operation (c/o) for ECALLS and OCALLS are significantly greater than a normal system call (7000c/o and 200c/o respectively). The bandwidth of allocating memory within the enclave was 30% compared to outside the enclave (4.0 GB/s outside the enclave, 1.2 GB/s inside the enclave).

The authors showed that allocating memory dynamically, inside the enclave, is significantly more expensive than outside the enclave. This issue was never explicitly mentioned in the developer guide by Intel [44] (see Section 2.12). The SGX developer guide [44] only states that this should be an issue if the enclave runs out of already allocated memory.

Zhao et al. [45] did not reveal why the performance overhead of allocating memory dynamically within the enclave is more expensive. However, the authors mention that the encryption of memory could cause the overhead.

3.4 VNF performance with SGX

Wang et al. [61] addressed how much SGX affects the performance of VNFs. The authors conducted three different experiments with different modes on the SGX: disabled, simulation mode, and hardware debug mode. Their experiments showed that using SGX in hardware debug mode added 176% additional latency compared to using no SGX (from approximately 31.3us to 86.3us), which the authors considered to be an acceptable level of overhead.

3.5 Provisioning secrets to VNFCs with SGX

Paladi and Karlsson [20] present a method for protecting VNFCs with integrity and confidentiality by using SGX. The authors also illustrate how TLS keys and certificates can be distributed to a VNF’s enclave from a remote server (verification manager), and the required steps to securely establish a TLS connection between a network controller and VNFC.

In the used method for distributing the TLS keys and certificates, the verification manager generates the keys and X.509 certificate, and sends them encrypted to the VNF’s enclave. The keys are known by the verification manager, which adds an unnecessary amount of attack surface towards the keys.

3.6 Protecting against isolation failure with SGX

Shih et al. [62] present a framework called S-NFV, which mitigates the isolation failure threat on a hypervisor by using SGX. The authors demonstrated their framework by securing tag operations in Snort [63] by running them in separate enclaves, or as the authors called them, S-NFV enclaves. As a result, even if the hypervisor is compromised, the tag operations are still protected because of the isolation protection provided by SGX.

3.7 vTPM issues addressed with SGX

Sun et al. [64] addressed the fundamental issues with a virtual TPM (vTPM) by using SGX to create an enclave that contained the same functionality as a TPM. One of the fundamental issues with vTPMs in cloud environments is that they can not be measured without breaking the chain of trust. Enclaves provide the functionality needed to measure itself and can, therefore, be used for addressing the measurement issues with vTPMs by creating an enclave that supports the same functionality as the TPM.
3.8 Trust in Telco clouds

Vignostad, Borger [65] investigated how trust can be established to the central domains of the NFV architecture framework. The author created an NFV architecture with OpenStack and performed bootstrap measurements with a TPM. Runtime measurements were collected by using Linux IMA with the help of SELinux. VNFs were protected by sealing information inside the MANO with measurements of OpenStack configuration files such that if any OpenStack configuration file was changed, the VNFs were unable to start.

3.9 Comparison of TEEs

Maene et al. [2] compared 12 different TEEs (SGX, Secureblue++, ARM TrustZone, Sanctum, ...) against seven security properties (isolation, memory protection, sealing, code confidentiality, ...), seven architectural features and three other important properties such as if the frameworks are open-source or academic.

None of the compared TEEs fulfilled all security properties. All of the compared TEEs were vulnerable for side-channel attacks except Sanctum and TXT & TPM (TXT used in combination with TPM). The top five TEEs with the most significant number of supported security properties are summarized in Table 3.1.

Memory was considered to be protected if the TEE was resistance to hardware attacks such as fault injection and probing. Only software-based side-channel attacks had to be mitigated for considering the TEE to support side-channel protection.

Table 3.1: The top five TEEs (if only supported security properties are considered) compared by Maene et al. [2].

<table>
<thead>
<tr>
<th>Property</th>
<th>TXT &amp; TPM</th>
<th>AEGIS</th>
<th>SGX</th>
<th>Sanctum</th>
<th>SecureBlue++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Attestation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sealing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Dynamic RoT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Code Confidentiality</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Side-channel Protection</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Memory Protection</td>
<td>/</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Note that even though the TXT & TPM supports the greatest number of security properties, it do not necessarily imply that they are a feasible solution for securing VNFs in cloud environments, because it is not feasible to use a TPM for measuring large number of dynamically allocated VM images in a cloud environment [12][65](see Section 2.9).
4 Method

The research questions could have been addressed by implementing the necessary protection to an existing VNF with Asylo and Open Enclave, and then test both implementations from a performance and security perspective. However, this approach limits the scope to one particular type of VNF. By addressing the research questions without specifying the VNF’s functionality, the results can be applied to any case where VNF’s are used.

Since the communication channels between VNFs and other parties can be secured with TLS 1.3, the next challenge is to distribute the X.509 certificate to VNFs without disclosing the certificate’s private key to an attacker. To verify that the necessary functionality for provisioning certificates is supported in Asylo and Open Enclave, a protocol for provisioning certificates to the enclaves needs to be addressed.

Paladi and Karlsson [20] presented a sequence of steps for distributing X.509 certificates to VNFs where the certificates and keys are generated inside the CA (referred to as the verification manager in the report by Paladi and Karlsson). This approach adds an additional amount of attack surface towards the private key because it is known by other parties than the certificate’s owner. Usually, the certificate is generated by the certificate’s owner along with the private key. The certificate is then sent to the CA, which is also referred to as a Certificate Signing Request (CSR). The CSR may be accompanied with additional information for authentication purposes. Therefore, the protocol presented by Paladi and Karlsson [20] is extended and improved in Section 4.1 such that the keys are generated inside the enclave and is never exposed to any other party than the enclave.

4.1 TEE X.509 Certificate Signing Protocol

The enclave initiates the protocol by generating an X.509 certificate and an asymmetric key-pair. The enclave then generates a remote attestation report and assigns the REPORTDATA field to an SHA-256 digest of the certificate. When the CA receives the remote attestation report and certificate, it can verify that the certificate originates to an enclave that can be trusted by examining the MRENCLAVE and MRSIGNER fields. The handshakes required between the enclave and CA is visualized in Figure 4.1. The certificate’s private key has never left the enclave, and the protocol used by Paladi and Karlsson [20] has therefore been improved from a security perspective. The communication between the enclave and CA is not encrypted because the exchanged information is not confidential. The signed certificate
that is received by the CA is protected against replay attacks since the public key inside the certificate can be viewed as a nonce (if the public key has already been used, somebody else knows the private key).

Remote attestation is usually initialized when the verifier generates a challenge (see Section 2.12) and sends it to the target to protect the attestation report from replay attacks. The certificate’s public key will provide replay protection from the enclave’s perspective, but the CA is vulnerable for receiving replayed CSRs for certificates with compromised public keys. However, if the CA never signs a certificate with a public key that has been seen before, replay attacks are mitigated.

A similar protocol is used in Open Enclave’s remote attestation example [67]. The REPORTDATA field is populated with a digested public key that is generated within the enclave. The remote attestation report and the public key is shared between the enclaves. If the attestation report can be successfully verified, and the public key matches the digested public key in the REPORTDATA field, the enclaves can start exchanging encrypted messages by using the public keys. The main difference between the protocol proposed in this section and the Open Enclave example is that an X.509 certificate is sent instead of a public key.

4.2 Security comparison

The impact on security is determined by the SDKs’ capability to extend the TEE hardware. The SDKs’ purpose is to abstract TEE hardware; they do not create any new security properties.

Since the impact on security is determined by the SDKs’ capabilities to extend the TEE hardware, the number of security properties supported through the SDKs can be used as a metric to determine the SDKs’ impact on security. However, all security properties may not be required for being able to provision certificates and protecting the VNFs during runtime. Therefore, a risk assessment is needed where the security properties can be mapped with countermeasures, such that the necessary security properties can be identified. The security properties considered are limited to the security properties used by Maene et al. [2]:

- **Isolation**: Applications can be isolated from other software that is running on the same host such that the application’s confidentiality and integrity are preserved.
- **Attestation**: Software that is running on the platform can be measured and verified remotely (remote attestation).
- **Sealing**: Secrets can be encrypted with a key that is dependent by the application’s measurement and hardware.
- **Dynamic RoT**: The RoT can be extended by dynamic software, e.g., user applications (see Section 2.9).
- **Code confidentiality**: Applications can be deployed and executed without revealing the application’s code.
4.2. Security comparison

- **Side-channel protection**: Applications are protected from software-based side-channel attacks.
- **Memory protection**: Applications are protected from hardware attacks, e.g., probing and fault injects.

The security properties used for mitigating the risks can be used for determining the most appropriate TEE hardware for protecting VNFs. If SGX is not the most appropriate hardware to use, then it suggests that Asylo and Open Enclave currently do not support the right hardware for protecting VNFs.

By combining the results from the risk assessment with the supported security properties in Open Enclave and Asylo, the impact on security from using the SDKs can be assessed.

Risk assessment

There are many different types of standardized methods that could be used for conducting a risk assessment. Most risk assessments share the following procedures: they identify the assets, vulnerabilities, threats, and risks. The risk assessment methodology was chosen based on its popularity and availability (should not be protected by payment wall). The following list summarizes two possible risk assessments that were considered during the thesis:

- The NIST SP 800-30 [58] and 800-39 [68] documents.
- The International Organization for Standardization and International Electrotechnical Commission (ISO/EIC) 27005 document [69]

NIST’s risk assessment extend the ISO/IEC 270005 such that organizations can follow both the NIST and ISO/EIC standards. NIST’s risk assessment was used because it extends the ISO/IEC 270005 and because of its availability compared to the ISO/EIC 270005, which was protected with a payment wall [69].

The NIST SP 800-30 [58] document presents a guide on how risk assessments should be conducted and can be summarized with the following steps:

- **Preparing for the assessment**: This step focuses on the assessment’s scope and purpose. The purpose of the risk assessment, in this thesis, is to identify the current threats against VNFs and the current vulnerabilities within VNFs. Therefore, no estimation of the risks is needed and was excluded from the risk assessment.

- **Conducting the assessment**: Threats, vulnerabilities, and the risks’ severity are identified. Since the risks are excluded from the assessment, only the threats and vulnerabilities are identified in this thesis during this phase. The vulnerability-oriented approach is used because there exists a large amount of related work that addresses current vulnerabilities within VNFs, and could be used in this risk assessment to identify vulnerabilities and threats.

- **Communicating the results**: The vulnerabilities and threats were presented in separate tables. The vulnerabilities were mapped with the attackers that could exploit the vulnerability. The threats were divided according to the attackers, e.g., the external threats table.

- **Maintaining the assessment**: This step addresses organizations need to keep the risk assessment updated when new threats and vulnerabilities appear. This step was ignored since future threats and vulnerabilities that may appear because of, e.g., changes in the environment is outside the scope of this thesis.

The NIST SP 800-39 [68] document presents a number of methods for mitigating risks and can be summarized with the following list:
4.2. Security comparison

- **Risk acceptance:** If the risk is considered to be tolerated, then the risk is accepted and no actions are taken to mitigate the risk.

- **Risk avoidance:** The activities that are causing the intolerable risks are avoided by not using the activities anymore.

- **Risk mitigation:** Countermeasures are implemented to protect the assets. This is the method used in this thesis to mitigate the risks.

- **Risk sharing or transfer:** The risks are transferred to another, e.g., organization.

**Environment**

The risk assessment extends the threats and attackers presented by Coppolino et al. [5] (internal, external and insider attackers) and the threats and best practices by Lal et al. [8], by including a CA into the environment (see Figure 4.2). As already mentioned in the introduction, the communication between VNFs is assumed to be using TLS 1.3. Therefore, the VNFs require a signed X.509 certificate by the CA for being able to establish secure communication channels with other VNFs. Certificates are signed according to the protocol from Section 4.1.

The VNFs establish a TLS connection with each other after the CA has successfully signed the VNF certificates. The attackers are assumed to become active directly when the VNF starts, which is when the VNF contacts the CA.

![Diagram of involved attackers, actors, and components](image)

**Figure 4.2:** Overview of the involved attackers, actors, and components. Communication between the VNFs is secured with TLS 1.3.

**Attackers**

The attackers are categorized according to the attack vectors presented by Coppolino et al. [5] i.e., internal, external, and insider attackers. If two attackers could not compromise an asset individually but can compromise it together, the attackers are allowed to collaborate. No research was conducted for finding collaborated attacks unless there were two attackers that could not compromise the same asset.

- **Internal attacker:** Deploys a malicious VNF and attacks the underlying infrastructure to gain control over the VNFs by using, e.g., a VM escape attack or side-channel at-
tack. The internal attacker was assumed to receive the necessary privileges to violate all assets (see next section) if the hypervisor was successfully compromised.

- **External attacker:** Performs network attacks, e.g., exploit implementation failures that are located within the VNFs by sending malicious network packets or tampers with the communication between VNFCs such that it benefits the attacker.

- **Insider attacker:** The insider attacker performs attacks that take advantage of its privileges as an employee. Probing, fault injection, and exploiting root privileges are examples of such attacks. The insider attacker was assumed to have the highest privileges possible on the host, and was therefore assumed to be capable of violating every asset (see next section).

### Assets

The assets were identified by first creating a list of the components that should be protected, i.e., the VNF and the X.509 certificate’s private key. The list was then expanded by considering every attribute of the CIA triad as an asset (see Table 4.1).

The meaning of asset AS3 can be interpreted in multiple ways. It was considered to be compromised if the certificate’s private key was either changed, deleted, or denied creation.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS1</td>
<td>Confidentiality of certificate’s private key</td>
</tr>
<tr>
<td>AS2</td>
<td>Integrity of certificate’s private key</td>
</tr>
<tr>
<td>AS3</td>
<td>Availability of certificate’s private key</td>
</tr>
<tr>
<td>AS4</td>
<td>VNF’s confidentiality</td>
</tr>
<tr>
<td>AS5</td>
<td>VNF’s integrity</td>
</tr>
<tr>
<td>AS6</td>
<td>VNF’s availability</td>
</tr>
</tbody>
</table>

### Approach

The vulnerability-oriented approach by NIST [58] was used for identifying threats. The vulnerabilities were identified by performing a literature study on available research about vulnerabilities in VNFs and cloud applications. Vulnerabilities that exist in VNFs exist in cloud applications as well, but not all vulnerabilities in cloud applications exist in VNFs. Therefore, vulnerabilities that were discovered from papers about cloud applications were only included in the risk assessment if it was trivial to understand that they existed in VNFs as well.

Vulnerabilities that was discovered during the literature study were mapped with attackers that can exploit the vulnerability, e.g., insider attackers only exploit vulnerabilities that are unique for employees at the CSP. Every vulnerability available for an attacker was considered as a possible weakness that could be exploited to compromise an asset (see Figure 4.3). If a vulnerability could be exploited to compromise an asset, a threat that summarized the attack was included to a list of threats posed by the attacker. For example, if the insider attacker can exploit three vulnerabilities, and there exist six assets, then there are 18 (3 x 6) insider threats if all vulnerabilities can be exploited.

### Countermeasures

A countermeasure either decreased the probability of a vulnerability being exploited or reduced the impact if a vulnerability was exploited. The countermeasures were identified by
4.3. Performance comparison

The goal was to capture the performance overhead of protecting VNFs with Open Enclave v0.5.0 and Asylo v0.3.4. If the SDKs were benchmarked similarly as Zhao et al. [45] by measuring primitives such as bandwidth and cycles per operation, the fastest and most memory efficient SDK could be determined. However, the results would be missing influences from production scenarios, and be harder to compare with real VNFs. Instead, the enclaves were performing a task that could be found in production scenarios, i.e., signing messages of different sizes with a key that has been provisioned to the enclave. By measuring the Round-Trip-Time (RTT) for signing a message inside the enclave, and comparing the results with experiments without TEE functionality, the performance penalty of using Open Enclave and Asylo can be determined.

Figure 4.3: All vulnerabilities available for an attacker were used for finding attacks against the assets. Note that vulnerabilities can be shared between many attackers (see VU06).

Using the security properties from Maene et al. [2] (see Section 3.9). A security property was considered to be necessary for protecting VNFs if it was used as a countermeasure for mitigating a threat.

Supported security properties

The security properties identified as necessary were identified in the SDKs by conducting a literature study on available documentation and examples about Open Enclave v0.5.0 and Asylo v0.3.4. A security property was assumed to be missing if it was not mentioned in the documentation or examples.

4.3 Performance comparison

The goal was to capture the performance overhead of protecting VNFs with Open Enclave v0.5.0 and Asylo v0.3.4. If the SDKs were benchmarked similarly as Zhao et al. [45] by measuring primitives such as bandwidth and cycles per operation, the fastest and most memory efficient SDK could be determined. However, the results would be missing influences from production scenarios, and be harder to compare with real VNFs. Instead, the enclaves were performing a task that could be found in production scenarios, i.e., signing messages of different sizes with a key that has been provisioned to the enclave. By measuring the Round-Trip-Time (RTT) for signing a message inside the enclave, and comparing the results with experiments without TEE functionality, the performance penalty of using Open Enclave and Asylo can be determined.
4.3. Performance comparison

Experiments

Open Enclave and Asylo offer different features that make them useful in different scenarios. Asylo supports functionality for communicating with enclaves remotely through gRPC. Open Enclave does not support remote communication with enclaves, which gives Asylo an advantage in scenarios where the TEE functionality is isolated in a separate container.

Isolating the TEE functionality in a separate container is good from a flexibility and scaling perspective, but contributes with a performance penalty (see Appendix A for a more thorough analysis). Therefore, it would be interesting to measure the performance overhead of isolating the TEE functionality in a separate container. The results will give the necessary information to discuss whether it is feasible from a performance perspective to isolate TEE functionality in a separate container.

If the same settings should be used as when the VNF is used in production, then a commercial license needs to be acquired (see Section 2.12), which was not considered as an option. Therefore, the SGX was used in debug mode for the experiments. The same experiments were also executed on a simulated SGX device for making it possible to compare the results with Wang et al. [61].

Experiments performed without gRPC communication are labeled SDK_local (e.g., asylo_local) and experiments with gRPC are labeled SDK_remote (e.g., asylo_remote). All experiments performed without TEE functionality is labeled no_tee.

No TEE functionality

The list of experiments for disclosing the relationship between packet sizes and RTTs without TEE functionality are visualized in Table 4.3.

Table 4.2: List of experiments for disclosing the relationship between packet sizes and RTTs without using TEE functionality.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>SDK</th>
<th>Separate Container</th>
<th>Inputs [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP01</td>
<td>no_tee</td>
<td>None</td>
<td>N</td>
<td>10^0, ..., 10^6</td>
</tr>
</tbody>
</table>

Simulated SGX

The list of experiments for disclosing the relationship between packet sizes and RTTs with a simulated SGX are visualized in Table 4.3. Note that the simulated SGX experiments are using the support for executing an SGX application without the hardware support.

Table 4.3: List of experiments for disclosing the relationship between packet sizes and RTTs with a simulated SGX.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>SDK</th>
<th>Separate Container</th>
<th>Inputs [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP02</td>
<td>oe_local</td>
<td>Open Enclave</td>
<td>N</td>
<td>10^0, ..., 10^6</td>
</tr>
<tr>
<td>EXP03</td>
<td>oe_remote</td>
<td>Open Enclave</td>
<td>Y</td>
<td>10^0, ..., 10^6</td>
</tr>
<tr>
<td>EXP04</td>
<td>asylo_local</td>
<td>Asylo</td>
<td>N</td>
<td>10^0, ..., 10^6</td>
</tr>
<tr>
<td>EXP05</td>
<td>asylo_remote</td>
<td>Asylo</td>
<td>Y</td>
<td>10^0, ..., 10^6</td>
</tr>
</tbody>
</table>
4.3. Performance comparison

Hardware SGX

The list of experiments for disclosing the relationship between packet sizes and RTTs with a hardware SGX in debug mode are visualized in Table 4.4.

Table 4.4: List of experiments for disclosing the relationship between packet sizes and RTTs with a hardware SGX in debug mode.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>SDK</th>
<th>Separate Container</th>
<th>Inputs [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP06</td>
<td>oe_local</td>
<td>Open Enclave</td>
<td>N</td>
<td>$10^0, ..., 10^6$</td>
</tr>
<tr>
<td>EXP07</td>
<td>oe_remote</td>
<td>Open Enclave</td>
<td>Y</td>
<td>$10^0, ..., 10^6$</td>
</tr>
<tr>
<td>EXP08</td>
<td>asylo_local</td>
<td>Asylo</td>
<td>N</td>
<td>$10^0, ..., 10^6$</td>
</tr>
<tr>
<td>EXP09</td>
<td>asylo_remote</td>
<td>Asylo</td>
<td>Y</td>
<td>$10^0, ..., 10^6$</td>
</tr>
</tbody>
</table>

Measurements

The implementations for recording local and remote RTTs were not identical because the remote measurements were performed in Python, and the local measurements were performed in C++. The local measurements are expected to be small for small packet sizes, which requires a measurement tool that can offer high resolution for small measurements. Botor et al. [70] compared the resolution of different measurement libraries such as `ctime` and `chrono`, and concluded that `ctime` should not be used for measurements smaller than 0.1ms. `chrono` performed well for small measurements and was therefore used for the local measurements. Listing 4.1 illustrates with pseudocode how the measurements were collected in C++ (local measurements).

```cpp
#include <chrono>

// Record timestamp before signature
start = std::chrono::high_resolution_clock::now()

// Sign packet within the enclave
signature = sign_packet( message )

// Record timestamp when signature is received
end = std::chrono::high_resolution_clock::now()

// Calculate the elapsed time in microseconds
RTT = chrono::duration_cast<chrono::microseconds>(end - start).count()

// Control that the signature was correct
if not correct_signature(message, signature): throw Error("Incorrect signature")
```

Listing 4.1: Pseudocode of how the local measurements were collected.

The remote measurements are expected to be higher than the local measurements and will not require the same resolution that was used for the local measurements. Therefore, the `time` library was used for the local measurements (see Listing 4.2).

```python
import time

// Get the timestamp before
start = time.time()
```
4.3. Performance comparison

Listing 4.2: Pseudocode of how the remote measurements were collected.

Enclave code

SGX requires an interface that defines ECALLs and OCALLs. Open Enclave only supported a certain amount of C data types for defining the interface towards the enclaves [71]. Asylo uses protobuf [72] for defining the enclave’s interface, which is a language-neutral mechanism from Google for serializing data. Strings are defined with the datatype char* in Open Enclave but were possible to be defined with the string datatype in Asylo.

The string datatype was not possible to be used for the interface in Open Enclave, and the most appropriate alternative to the char* datatype in protobuf was considered to be the string datatype. Therefore, the code within the Asylo enclaves signed messages of the type string, and the Open Enclave implementation signed messages of the type char*.

Open Enclave Code

The pseudocode presented in Listing 4.3 was used in the performance experiments for signing a message within an enclave in Open Enclave. The oe_host_malloc function performs an OCALL to allocate memory outside the enclave such that the signature can be accessed by the host [73].

```c
const char* secret_key = "secret_key";

int sign_data(const char* msg, size_t msg_size, char** signature, size_t* signature_size) {
    // 1. Allocate the necessary memory for the message to sign + the size of secret key.
    char* prepared_msg = (char*)malloc(sizeof(char) * (strlen(msg) + strlen(secret_key)));

    // 2. Copy the message to sign into the allocated memory
    strcpy(prepared_msg, msg);

    // 3. Concatenate the secret key with the message
    strncat(prepared_msg, secret_key, strlen(secret_key));

    // 4. Calculate MAC (Message Authentication Code) with memory that can be accessed by the host
    *signature = (char*)oe_host_malloc(sizeof(char) * 64);

    // 5. Generate SHA-256 digest
    sha256(*signature, prepared_msg);

    // 6. Free enclave memory
    free(prepared_msg);

    // 7. Set the size of the signature
    *signature_size = strlen(*signature);
}```
4.3. Performance comparison

Listing 4.3: Pseudocode of generating a HMAC with Open Enclave.

Step one, two and three in Listing 4.3 were supposed to perform the same amount of work as the `+` operator in `string`, and was expected to not contribute with any significant difference in performance.

The Open Enclave GitHub branch with the tag v0.5.0 was used because it was considered to be the latest and most up-to-date version when the performance experiments were performed on Open Enclave.

Asylo Code

The pseudocode presented in Listing 4.4 was used in the performance experiments for signing a message within an enclave in Asylo. The `GetMessage` and `SetSignature` functions perform the necessary operations for being able to read arguments from the host and write data back to the host.

```cpp
std::string secret_key = "secret_key";

Status Run(const EnclaveInput &input, EnclaveOutput *output) {
    // 1. The string is read from the input buffer
    std::string plaintext = GetMessage(input);
    // 2. The '+' operator is used and should, therefore, result in a string with
    // a newly allocated memory with the secret key concatenated
    std::string prepared_message = plaintext + secret_key;
    // 3. The message concatenated with the secret key is signed with the SHA-256
    // hash function. The HMAC is then assigned to the output buffer such that
    // the host can access the signature
    SetSignature(output, sha256(prepared_message));
    // 4. The enclave exits with status OK
    return Status::OkStatus();
}
```

Listing 4.4: Pseudocode of generating a HMAC with Asylo.

The v0.3.4-hotfix GitHub branch was used because it was considered to be the latest and most up-to-date version when the performance experiments were performed on Asylo.

No TEE Code

The pseudocode presented in Listing 4.5 was used in the performance experiments for signing a message without TEE functionality. The implementation is almost identical with the Open Enclave implementation, except that the signature was returned instead of assigned to a pointer.

```c
const char* secret_key = "secret_key";

char* sign_message(char* msg) {
    char* prepared_msg = (char*)malloc(sizeof(char) * (strlen(msg) + strlen(secret_key)));
    strcpy(prepared_msg, msg);
    strncat(prepared_msg, secret_key, strlen(secret_key));
    // Calculate MAC (Message Authentication Code)
    char* signature = (char*)malloc(sizeof(char) * 64);
```
4.3. Performance comparison

```c
sha256(signature, prepared_msg);
free(prepared_msg);
return signature;
```

Listing 4.5: Pseudocode of generating a HMAC with no TEE functionality.

**Data processing**

Figure 4.4 illustrates how the experiments were conducted. A sample represents an RTT measurement for signing a message inside the enclave. 200 samples are collected for each packet size until 200 samples of 1Mb have been collected. The same procedure is then repeated 25 times until each packet size array contains 5000 (200 x 25) samples. The enclave was kept alive during the entire experiment for making it possible to observe how the enclave behaves when it has been running for a longer amount of time (memory leakage and unexpected behaviors can be detected).

**Environment**

The experiments were performed on the machine described in Table 4.5. All experiments were executed inside a Docker container for making it possible to perform the experiments on another machine without reinstalling the dependencies (if necessary).

The remote experiments were tested with two containers, one with the gRPC server and enclave functionality, and another one that sent the gRPC calls and measured the RTT. The client was implemented in Python to minimize the necessary amount of work to create the client.
### Table 4.5: Summary of the machine that was used for the experiments.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Ubuntu 18.04 LTS (bionic)</td>
</tr>
<tr>
<td>Device</td>
<td>server</td>
</tr>
<tr>
<td>Kernel</td>
<td>Linux 4.15.0-48-generic</td>
</tr>
<tr>
<td>System</td>
<td>Dell</td>
</tr>
<tr>
<td>Product</td>
<td>PowerEdge R230</td>
</tr>
<tr>
<td>CPU</td>
<td>Quad core Intel Xeon E3-1270 v6 (-MT-MCP-)</td>
</tr>
<tr>
<td>Cache</td>
<td>8192 KB</td>
</tr>
<tr>
<td>Flexible Launch Control</td>
<td>Not supported</td>
</tr>
<tr>
<td>SGX 1.0</td>
<td>Supported</td>
</tr>
<tr>
<td>SGX 2.0</td>
<td>Not supported</td>
</tr>
<tr>
<td>Docker Version</td>
<td>18.09.05</td>
</tr>
<tr>
<td>Python Version</td>
<td>3.6.7</td>
</tr>
</tbody>
</table>
The results from performing the methodology in Chapter 4 are presented in this chapter.

5.1 Risk assessment

This section presents the results of the risk assessment. Each attacker could compromise every asset and is the reason why there is no section about collaborated attacks (multiple attackers working together to compromise an asset).

Vulnerabilities

Table 5.1 summarize the discovered vulnerabilities from the literature study:

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Attacker</th>
</tr>
</thead>
<tbody>
<tr>
<td>VU01</td>
<td>Vulnerabilities in the hypervisor can be exploited by an attacker to perform a VM escape attack [8][5]. If the attacker successfully escapes the VM, it has enough privileges to compromise the host and neighbor tenants [78].</td>
<td>Internal</td>
</tr>
<tr>
<td>VU02</td>
<td>Vulnerabilities that are located within the VNFs can be exploited through the VNF’s network connection to perform, e.g., remote code execution attacks or SQL injections [5].</td>
<td>External</td>
</tr>
<tr>
<td>VU03</td>
<td>Tenants share the same hardware resources and can be exploited by an attacker to extract sensitive data by performing, e.g., a side-channel attack. [5]</td>
<td>Internal</td>
</tr>
<tr>
<td>VU04</td>
<td>All hardware resources are controlled by the CSP, and can therefore be altered by an attacker with the purpose to change or extract secrets. [5][8]</td>
<td>Insider</td>
</tr>
<tr>
<td>VU05</td>
<td>Communication channels on the network layer can be intercepted, changed and replied by an attacker. [5]</td>
<td>External</td>
</tr>
<tr>
<td>VU06</td>
<td>All hardware resources can be allocated by an attacker and is therefore vulnerable for Denial-of-Service (DoS) attacks. [8][79]</td>
<td>Internal</td>
</tr>
</tbody>
</table>
5.1. Risk assessment

Tenants cannot control where the VNF is deployed, and is therefore vulnerable for attacks where the VNF is deployed in a country that violates regulatory compliances [8], e.g., GDPR.

The VNF’s secrets are not protected and could be altered without possessing the same privileges as the VNF by, e.g., generating a backup of the VMs memory [8] or by installing malicious software that extracts the secrets [5][7][9].

The underlying infrastructure (e.g., hardware and memory) are controlled by the CSP and can be misused by employees at the CSP to perform, e.g., DoS attack.

The VNF’s code is not protected and could be altered without possessing the same privileges as the VNF by, e.g., generating a backup of the VM image.

External threats

Threats posed by the external attacker are presented in Table 5.2. Note that the vulnerabilities VU08 and VU10 were not exploited to compromise any asset because if an implementation failure is exploited, it is assumed that the attacker can execute code with the same privileges as the VNF.

Table 5.2: Threats posed by the external attacker.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Vuln.</th>
<th>Asset</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT1</td>
<td>The external attacker exploits a vulnerability in the VNF and injects code that extracts the VNF’s private key.</td>
<td>VU02</td>
<td>AS1</td>
</tr>
<tr>
<td>EXT2</td>
<td>The external attacker exploits a vulnerability in the VNF and injects code that changes the VNF’s private key.</td>
<td>VU02</td>
<td>AS2</td>
</tr>
<tr>
<td>EXT3</td>
<td>The communication between the CA and VNF is hijacked with a MITM-attack, and all signatures produced by the CA is changed such that the VNF will be denied TLS communication.</td>
<td>VU05</td>
<td>AS3</td>
</tr>
<tr>
<td>EXT4</td>
<td>The external attacker exploits a vulnerability in the VNF and injects code that deletes the VNF’s private keys.</td>
<td>VU02</td>
<td>AS3</td>
</tr>
<tr>
<td>EXT5</td>
<td>The external attacker exploits a vulnerability in the VNF and injects code that reveals the VNF’s code.</td>
<td>VU02</td>
<td>AS4</td>
</tr>
<tr>
<td>EXT6</td>
<td>The external attacker exploits a vulnerability in the VNF and injects code that terminates the currently running VNF and launches a new malicious VNF with the same credentials.</td>
<td>VU02</td>
<td>AS5</td>
</tr>
<tr>
<td>EXT7</td>
<td>The external attacker exploits a vulnerability in the VNF and injects code that terminates the VNF.</td>
<td>VU02</td>
<td>AS6</td>
</tr>
<tr>
<td>EXT8</td>
<td>The external attacker performs a MITM-attack and blocks messages from the VNF.</td>
<td>VU05</td>
<td>AS6</td>
</tr>
</tbody>
</table>
### Internal threats

Threats posed by the internal attacker are presented in Table 5.3.

Table 5.3: Threats posed by the internal attacker.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Vuln.</th>
<th>Asset</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT1</td>
<td>The internal attacker performs a VM escape attack and extracts the VNF’s private key.</td>
<td>VU01, VU08</td>
<td>AS1</td>
</tr>
<tr>
<td>INT2</td>
<td>The internal attacker performs a side-channel attack and extracts the VNF’s private key.</td>
<td>VU03, VU08</td>
<td>AS1</td>
</tr>
<tr>
<td>INT3</td>
<td>The internal attacker performs a VM escape attack and changes the VNF’s private key.</td>
<td>VU01, VU08</td>
<td>AS2</td>
</tr>
<tr>
<td>INT4</td>
<td>The internal attacker performs a VM escape attack and deletes the VNF’s private key.</td>
<td>VU01, VU08</td>
<td>AS3</td>
</tr>
<tr>
<td>INT5</td>
<td>The internal attacker performs a VM escape attack and extracts the VNF’s code.</td>
<td>VU01, VU10</td>
<td>AS4</td>
</tr>
<tr>
<td>INT6</td>
<td>The internal attacker performs a VM escape attack and replaces the victims VNF with a malicious VNF that has access to the same credentials as the original VNF.</td>
<td>VU01, VU08, VU10</td>
<td>AS5</td>
</tr>
<tr>
<td>INT7</td>
<td>The internal attacker performs a VM escape attack and terminates the VNF.</td>
<td>VU01</td>
<td>AS6</td>
</tr>
<tr>
<td>INT8</td>
<td>The internal attacker allocates all available resources to perform a DoS attack against the VNF.</td>
<td>VU06</td>
<td>AS6</td>
</tr>
</tbody>
</table>

### Insider threats

Threats posed by the insider attacker are presented in Table 5.4.

Table 5.4: Threats posed by the insider attacker.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Vuln.</th>
<th>Asset</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS1</td>
<td>The insider attacker generates backup of the VM and extracts the VNF’s private key.</td>
<td>VU04, VU08</td>
<td>AS1</td>
</tr>
<tr>
<td>INS2</td>
<td>The insider attacker uses root privileges to replace the VNF with an identical VNF that has another certificate.</td>
<td>VU04, VU08</td>
<td>AS2</td>
</tr>
<tr>
<td>INS3</td>
<td>The insider attacker uses root privileges to delete the VNF’s private key.</td>
<td>VU04, VU08</td>
<td>AS3</td>
</tr>
<tr>
<td>INS4</td>
<td>The VNF’s code is revealed by generating a backup of the VNF’s image.</td>
<td>VU04, VU10</td>
<td>AS4</td>
</tr>
<tr>
<td>INS5</td>
<td>The insider attacker replaces the VNF with a malicious VNF that has the same certificate.</td>
<td>VU04, VU08, VU10</td>
<td>AS5</td>
</tr>
<tr>
<td>INS6</td>
<td>The insider attacker launches the VNF in a geographical location where the VNF violates regulatory compliance.</td>
<td>VU07</td>
<td>AS5</td>
</tr>
<tr>
<td>INS7</td>
<td>The insider attacker uses root privileges to terminate the VNF’s VM.</td>
<td>VU04</td>
<td>AS6</td>
</tr>
<tr>
<td>INS8</td>
<td>The available resources are limited by the insider attacker.</td>
<td>VU09</td>
<td>AS6</td>
</tr>
<tr>
<td>INS9</td>
<td>The insider attacker launches the VNF on a geographical location such that the latency of communicating with the VNF is unacceptable.</td>
<td>VU07</td>
<td>AS6</td>
</tr>
</tbody>
</table>
5.2 Risk response

This section presents a number of countermeasures that mitigates the threats from Section 5.1. The countermeasures are implemented by introducing two additional components, the Secure-VNFC (S-VNFC) and Certificate Authority-VNFC (CA-VNFC). The S-VNFC and CA-VNFC are described further in Sections 5.2 and 5.2. Conclusions about mitigated threats and vulnerabilities that could not be mitigated by the use of TEE technology is discussed further in Chapter 6.

Isolating sensitive functionality

Code that should be protected is isolated into a S-VNFC, which is a separate VNFC dedicated for protecting functionality with TEE hardware. The isolated functionality can be reached over a network protocol e.g., gRPC. Each S-VNFC generates a X.509 certificate that needs to be signed by either the CA or CA-VNFC (see Section 5.2). The protocol for signing the certificate is described in Section 4.1. The necessary security properties for providing the maximum level of isolation are summarized in Table 5.5.

<table>
<thead>
<tr>
<th>Vuln.</th>
<th>Motivation</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>VU01</td>
<td>By using the TEE’s isolation property, an additional layer is added between the hypervisor and neighbor tenants. The risk of isolation failure has, therefore, been mitigated (see Section 3.6).</td>
<td>Isolation</td>
</tr>
<tr>
<td>VU02</td>
<td>The sensitive functionality is isolated into S-VNFCs, and for being able to exploit a vulnerability inside the isolated functionality, the vulnerability needs to be located within the isolated code. The attack surface has therefore been reduced, and the vulnerability is considered to be mitigated.</td>
<td>Isolation</td>
</tr>
<tr>
<td>VU03</td>
<td>Software-based side-channel attacks are mitigated by using the TEE’s side-channel protection.</td>
<td>Side-channel Protection</td>
</tr>
<tr>
<td>VU04</td>
<td>Memory used by the isolated functionality is encrypted, and not controlled by the CSP.</td>
<td>Memory Protection</td>
</tr>
<tr>
<td>VU08</td>
<td>The secrets are encrypted in the memory, which reduces the impact of e.g., a VM backup.</td>
<td>Memory Protection</td>
</tr>
</tbody>
</table>

The vulnerability VU10 is not included in Table 5.5 because the enclave could still be vulnerable for code theft during deployment. However, the code’s confidentiality will be protected during runtime but was not considered as enough for mitigating vulnerability VU10.
5.2. Risk response

Code confidentiality protection

Code confidentiality is protected by using the same method that was used by Lazard et al. \[60\] in TEEshift. The key used for decrypting the code needs to be provisioned to the enclave, and can be integrated with the protocol from Section 4.1.

Figure 5.1: The VNF’s decryption key is accompanied with the signed certificate.

Figure 5.1 illustrates how the decryption key is added to the CA’s reply. The reply is encrypted with the public key that was included with the CSR, in case that the same decryption key is used for multiple VNFs. The decryption key can be provisioned to the CA in a number of ways, but is considered to be outside the scope of this thesis and is therefore not described further.

Table 5.6: Vulnerabilities mitigated by using the proposed code confidentiality protection.

<table>
<thead>
<tr>
<th>Vuln.</th>
<th>Motivation</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>VU10</td>
<td>By utilizing the TEE’s code confidentiality, attestation, and dynamic RoT security properties, code can be deployed encrypted and the impact if the code is revealed has been reduced. The dynamic RoT is needed because the deployed application’s software would otherwise not be measurable.</td>
<td>Attestation, Code Confidentiality, Dynamic RoT</td>
</tr>
</tbody>
</table>

Local certificate distribution

The CA can sign certificates for each S-VNFC or for one S-VNFC that is responsible for signing certificates to other S-VNFCs. The first alternative is exhaustive for the CA, and if the connection to the CA for some reason would be compromised, then no new S-VNFCs could be initialized. Therefore, each VNF can be equipped with a dedicated S-VNFC for signing certificates (CA-VNFC) to other S-VNFCs.

The proposed protocol in Section 4.1 is used for provisioning a valid certificate to the CA-VNFC. The S-VNFC is also using the same protocol for receiving signed certificates by the CA-VNFC. The mitigated vulnerabilities are summarized in Table 5.7.

Table 5.7: Vulnerabilities mitigated by using the proposed local certificate distribution protocol.

<table>
<thead>
<tr>
<th>Vuln.</th>
<th>Motivation</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>VU05</td>
<td>By utilizing the attestation and dynamic RoT support, the attack surface for performing DoS attacks against the communication with the CA has been reduced.</td>
<td>Attestation, Dynamic RoT</td>
</tr>
</tbody>
</table>

Regulatory compliance protection

Section 5.2 proposed a method for deploying VNFs without disclosing the VNF’s protected functionality. If the key, however, is not sent to the enclave, it can not be launched since the
5.3. Supported security properties

This section presents the supported security properties in Open Enclave and Asylo. Table 5.9 presents the security properties used by the countermeasures in Section 5.2 (see tables 5.6, 5.7 and 5.8), and supported security properties in the SDKs. Neither Open Enclave or Asylo supported functionality for deploying applications with code confidentiality protection. Asylo was also missing support for remote attestation. Isolation, memory protection, dynamic RoT, and side-channel protection are supported by the TEE hardware and do not need to be exposed through the SDKs. Isolation, memory protection, and dynamic RoT were marked as supported because they are supported in SGX. Code confidentiality protection requires a key from the SDK’s user for encrypting the software. Therefore, code confidentiality was included to the literature study about supported security properties in Open Enclave and Asylo (see results in Sections 5.3 and 5.3).

Table 5.9: Supported functionality in Open Enclave and Asylo.

<table>
<thead>
<tr>
<th>Property</th>
<th>SGX</th>
<th>Open Enclave v0.5.0</th>
<th>Asylo v0.3.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start &amp; Terminate</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Isolation</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Memory protection</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Side-channel protection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code confidentiality</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote attestation</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic RoT</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Supported security properties in Open Enclave

This section describes how the security properties are supported in Open Enclave (see Table 5.9).
5.3. Supported security properties

**Start & Terminate**

Enclaves are started in Open Enclave by executing a start function generated by the `oeedger8r` tool. The samples provided with the Open Enclave SDK starts and terminates an enclave, and was therefore used for verifying that the start and terminate enclave functionality is supported. The example was built and executed inside a Docker container with a simulated SGX device.[80]

![Figure 5.2: Screenshot of running Open Enclave's helloworld example.](image)

**Code confidentiality**

No examples or documentation were found about supported code confidentiality functionality in Open Enclave.

**Remote attestation**

The `oe_get_report` function is used for generating local and remote attestation reports.[81] The `oe_get_report` function takes a flag as argument that instructs Open Enclave to either produce a local or remote attestation report.[81] Attestation reports are wrapped inside a `oe_report_header_t` structure which contains a field that describes the report’s type i.e., local or remote.[81] The `oe_verify_report` function can decide how to verify a report by looking at this field.

The remote attestation example could not be executed because the hardware that was available for this thesis did not support Flexible Launch Control (FLC).

**Supported security properties in Asylo**

This section describes how the SGX functionality is supported in Asylo (see Table 5.9). All functionality exposed to the developer for running enclaves is located in the `EnclaveClient` class, which is an abstract class extended by each TEE hardware (e.g., SGX).[75]

**Start & Terminate**

Asylo’s `quickstart` example starts and terminates an enclave with the `EnterAndRun` and `DestroyEnclave` functions. Starting and terminating an enclave was therefore marked as supported.[75]

![Figure 5.3: Screenshot of running Asylo’s quickstart example.](image)
5.4. Performance experiments

Code confidentiality

No examples or documentation were found about supported code confidentiality functionality in Asylo. Lazard et al. [60] encrypted the binary after it had been compiled, which was not necessary if Asylo already provided code confidentiality.

Remote attestation

No examples or documentation about how to perform remote attestation in Asylo was found during the literature study.

5.4 Performance experiments

The results from experiments EXP01 - EXP09 are presented in Section 5.4 and 5.4. The same no_tee results were used in Figure 5.4 and Figure 5.7.

Simulated SGX

The results from the performance experiments with a simulated SGX are visualized in Figure 5.4. Each bar is a mean value of 5000 samples for a specific packet size. The thin line within the bars is the standard deviation of all 5000 samples. The scatter plots (see figures 5.5 and 5.6) visualizes the raw data that is plotted in Figure 5.4.
5.4. Performance experiments

Figure 5.5: Scatter plot of remote Asylo and Open Enclave samples with a simulated SGX.

Figure 5.6: Scatter plot of local Asylo and Open Enclave samples with a simulated SGX.
5.4. Performance experiments

Hardware SGX

The results from the performance experiments with a hardware SGX in debug mode are visualized in Figure 5.7. Each bar is a mean value of 5000 samples for a specific packet size. The thin line within the bars is the standard deviation of all 5000 samples. The scatter plots (see figures 5.8 and 5.9) visualizes the raw data that is plotted in Figure 5.7.

Figure 5.7: Results of performance experiments with a hardware SGX in debug mode.

Figure 5.8: Scatter plot of remote Asylo and Open Enclave samples with a hardware SGX in debug mode.
Figure 5.9: Scatter plot of local Asylo and Open Enclave samples with a hardware SGX in debug mode.
This chapter presents an analysis of the results (see Section 6.1), and the used methodology for producing the results (see Section 6.2).

### 6.1 Result

This chapter discusses the results by analyzing them further and comparing them with related work.

#### Security comparison

The LCM operations presented by ETSI were not included to the risk assessment, and there is a possibility that some of the security properties that were not considered as necessary (sealing, local attestation, ...) could be useful for, e.g., migrating a VNF to another platform. Therefore, investigating how LCM operations can be protected with TEE hardware is added to the list of future work (see Section 7.3).

#### Appropriate TEE hardware for protecting VNFs

None of the TEEs in Maene et al. [2] supported all the necessary functionality to implement the risk responses from Section 5.2. The TXT & TPM was missing the memory protection property (is only partially supported). If the memory protection property were entirely supported, every required security property would have been supported by using TXT & TPM. However, the TXT & TPM was excluded as a possible candidate to secure VNFs because the TXT interrupts all services on a machine during execution and TPMs are not efficient in virtualized environments [12][66].

SecureBlue++ do not support attestation and can, therefore, be disregarded as a candidate for protecting VNFs, which implies that SGX, AEGIS, and Sanctum are the candidates left for protecting VNFs. The difference between SGX, AEGIS, and Sanctum (if only Table 3.1 is considered) is that Sanctum supports side-channel protection but has no memory protection. SGX and AEGIS support memory protection but is vulnerable to side-channel attacks. According to Table 6.1, memory protection mitigated more threats than side-channel protection. Sanctum is therefore disregarded as a suitable TEE for protecting VNFs.
According to Maene et al. [2], AEGIS and SGX support the same security properties and is therefore not analyzed further because it can be concluded that Asylo and Open Enclave supports a promising TEE hardware for securing VNFs.

Table 6.1: Summary of mitigated threats and vulnerabilities if SGX is used.

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Threats</th>
<th>Mitigated</th>
</tr>
</thead>
<tbody>
<tr>
<td>VUL08</td>
<td>9</td>
<td>X</td>
</tr>
<tr>
<td>VUL01</td>
<td>6</td>
<td>X</td>
</tr>
<tr>
<td>VUL02</td>
<td>6</td>
<td>X</td>
</tr>
<tr>
<td>VUL04</td>
<td>6</td>
<td>X</td>
</tr>
<tr>
<td>VUL10</td>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>VUL05</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>VUL07</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>VUL03</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>VUL06</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>VUL09</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

A summary of the mitigated vulnerabilities and threats when SGX is used is visualized in Table 6.1. All remaining threats are related to either side-channel or DoS attacks.

Supported security properties

Asylo did not support remote attestation when this thesis was conducted, which has a large impact on security because remote attestation made it possible to establish trust to VNFs.

Neither Asylo or Open Enclave supported functionality for deploying applications with code confidentiality. However, code confidentiality can still be protected by using the method proposed by Lazard et al. [60] (see Section 3.2).

Performance comparison

Asylo outperformed Open Enclave in all performance experiments when a simulated SGX was used. The implementations were however not identical, which could have affected the results. The Open Enclave experiments should be slightly faster because the string library used within the enclave in the Asylo experiments should cause an overhead from, e.g., keeping track of the string’s total length. This was also verified by executing an experiment with the code illustrated in Listing 6.1 and 6.2.

```c
const char* secret_key = "secret_key";

// 1. Allocate the necessary memory for the message to sign + the size of secret key.
char* prepared_msg = (char*)malloc(sizeof(char) * (strlen(msg) + strlen(secret_key)));

// 2. Copy the message to sign into the allocated memory
strcpy(prepared_msg, msg);

// 3. Concatenate the secret key with the message
strncat(prepared_msg, secret_key, strlen(secret_key));
```

Listing 6.1: The string copy and concatenation operations performed within the Open Enclave implementation.

The msg variable was 1MB and the experiments were executed on an Ubuntu 18.04 without using any TEE functionality. The execution time of measuring the code in Listing 6.1 was 90µs compared to approximately 200µs with the code in Listing 6.2. Therefore, it can be
concluded that the implementations should not cause a large difference in performance, and if the performance penalty caused by the SDKs were identical, the Open Enclave experiments should be slightly faster.

```cpp
const char* secret_key = "secret_key";
std::string prepared_msg = msg + secret_key;
```

Listing 6.2: The string copy and concatenation operations performed within the Asylo implementation.

**Standard deviation**

The standard deviation of remote measurements was generally larger than the local measurements (see figures 5.5, 5.6, 5.8 and 5.9), which was expected because more noise can interfere with the experiments during remote measurements compared to local measurements.

Asylo’s standard deviation of local and remote measurements is higher than Open Enclave on many occasions (see figures 5.5 and 5.6). The reason that causes this behavior was never discovered in this thesis. However, meanwhile the experiments for a simulated SGX were running, it was noticed that Asylo’s RTTs were increasing with time (see Figure 6.1). The same behavior was not observed with Open Enclave (see Figure 6.2), and suggests that there is some problem with either the Asylo experiments or the currently used version of Asylo (e.g., memory leakage).

![Figure 6.1: Scatter plot of 5000 Asylo measurements with a simulated SGX.](image)

**Latency of using Open Enclave and Asylo**

The performance overhead (latency) of using a simulated SGX and hardware SGX, compared to using no_tee, is visualized in figures 6.3 and 6.4. The following equation describes how latency was calculated:

\[
\text{Latency} = 100 \cdot \left( \frac{\text{meas}}{\text{no_tee}} - 1 \right)
\]  

The `meas` and `no_tee` parameters are an average of 5000 RTT samples of the same packet size. The `no_tee` measurements can be found in Figure 5.4 and Figure 5.7 (the yellow bars).
6.1. Result

The latency of using an SGX, compared to no SGX, comprises the performance penalty of performing an ECALL and allocating memory within the enclaves [45]. For small packets, the memory allocation and execution time of signing a message are redundant. The latency can, therefore, be described by Equation 6.1 with the following assumption:

\[ \text{meas} = \text{no_tee} + \Delta \text{meas} \]  
(6.2)

where \( \Delta \text{meas} \) is the average performance overhead of performing a single ECALL. Equation 6.2 is combined with Equation 6.1 in the following equation:

\[ \text{Latency(meas)} = 100 \cdot \left( \frac{\text{no_tee} + \Delta \text{meas}}{\text{no_tee}} - 1 \right) = \frac{100 \cdot \Delta \text{meas}}{\text{no_tee}} \]  
(6.3)

The no_tee measurements are small for small packet sizes, which explains the high latency for small packet sizes in figures 6.3 and 6.4:

\[ \lim_{\text{no_tee} \to 0} \text{Latency(meas)} = \lim_{\text{no_tee} \to 0} \frac{100 \cdot \Delta \text{meas}}{\text{no_tee}} = \infty \]
6.1. Result

Figure 6.4: Latency with a hardware SGX in debug mode.

If the performance overhead of using Open Enclave and Asylo were constant for all packet sizes, the latency can be described with Equation 6.3. For large no_tee measurements, the latency should be approaching zero,

\[
\lim_{\text{no}_\text{tee} \to \infty} \text{Latency}(\text{meas}) = \lim_{\text{no}_\text{tee} \to \infty} \frac{100 \cdot \Delta \text{meas}}{\text{no}_\text{tee}} = 0
\]

which cannot be identified in figures 6.3 and 6.4. Therefore, it can be concluded that the ECALL’s performance overhead is not constant for different packet sizes, and the performance overhead of operations performed from within the enclave (e.g., allocating memory) is more expensive than outside the enclave since the latency plots’ derivatives are approaching zero for larger packet sizes (see figures 6.3 and 6.4).

Comparing performance results with related work

Wang et al. [61] used different packet rates (frequency of ECALLs) for measuring the performance penalties of protecting VNFs with SGX. The authors concluded that packet rates have a low impact on performance for all SGX modes (simulated, hardware in debug mode, ...). The average execution time of experiments with no SGX was approximately 31.3µs. The latency of using a simulated SGX and hardware SGX in debug mode compared to no SGX were 62% and 176% respectively.[61]

An execution time of 31.3µs with no_tee implicates that the packet size is somewhere between 1 and 100 bytes (see Figure 5.4 and Figure 5.7). The authors did not perform any remote measurements, which means only the latency of local measurements should be considered when comparing the results between this thesis and Wang et al. [61]. The performance overhead with Open Enclave and Asylo are a lot higher (1174% and 741% for one byte) if only the latency of local measurements is considered.

The authors mentioned that certain implementations caused a latency that was more than ten times compared to using no SGX [61], which suggests that the authors experienced similar overhead as when small packet sizes were used in figures 6.3 and 6.4. There are several explanations for why the latency in this thesis is higher than the latency from Wang et al. [61]:

- The size of arguments entered to the enclaves could affect the execution time of ECALLs.
- Open Enclave and Asylo should cause an additional performance penalty.
6.2 Method

The debug mode’s impact on performance

A commercial license was not used. According to Intel’s documentation \cite{41}, enclaves that are running in release mode or pre-release mode is suitable for performance experiments (see Section 2.12). It was never determined whether Open Enclave and Asylo generated enclaves in debug mode or pre-release mode. If the enclaves were not running in pre-release mode, it could have affected how realistic the results are compared with experiments where enclaves are running in release mode, since the compiler optimizations could have increased the performance of both SDKs. If only one SDK was running in pre-release mode, then it also could have affected the difference in performance between Asylo and Open Enclave.

Simulated SGX

The experiments with a simulated SGX were performed to generate more results that can be verified with the performance experiments from Wang et al. \cite{61}. Experiments with a hardware SGX reveal more information about how the VNFs would behave in a production scenario than experiments with a simulated SGX. However, performing experiments with a simulated SGX does not require any particular hardware and is, therefore, easier to perform. Future researchers that only wants to perform experiments with a simulated SGX can use the results from this thesis to predict how much larger the performance penalties would have been if a hardware SGX were used.

The \textit{asylo\_local} measurements for packet sizes larger than 10kb was smaller than the recorded measurements for \textit{no\_tee} (see Figure 5.4), which was unexpected because there should be an additional overhead of running an ECALL compared to a function call \cite{45}. This error also propagated to the latency plot with a simulated SGX, and is the reason why the \textit{asylo\_local} plot disappears after 10kb (see Figure 6.3).

Remote measurements with small packet sizes caused latencies of 55942\% and 40886\% for Open Enclave and Asylo (see Figure 6.3). For large packet sizes, both Open Enclave and Asylo had a lower latency with a simulated SGX compared to when a hardware SGX was used: 222\% and 170\% compared to 223\% and 279\% with a hardware SGX in debug mode.

Hardware SGX

Wang et al. \cite{61} showed that a hardware SGX has a higher performance penalty than a simulated SGX, and is supported by the results from this thesis in figure 5.4 and 5.7. The standard deviation of experiments with Asylo was smaller with a hardware SGX compared to when a simulated SGX was used. Therefore, it can be concluded that the simulated SGX had a negative effect on Asylo’s standard deviation.

The relative performance penalties from communicating with the TEE functionality remotely with small packet sizes was very high (see figures 6.3 and 6.4). More precisely, the latencies were 56752\% and 38706\% respectively. However, when the workload inside the enclave was large (1Mb), the latencies were 223\% and 279\% for Open Enclave and Asylo. As a conclusion, the proposed S-VNFC component in Section 5.2 would cause a significant performance penalty for small workloads. Depending on the context, the performance penalties could be acceptable for larger workloads within the enclave.

6.2 Method

Research questions Q1 and Q2 are quite general and was, therefore, dissembled into smaller parts that were considered to be the most interesting aspects to consider during the time the thesis was conducted.
6.2. Method

Security comparison

The number of supported security properties was used as a metric for measuring the impact on security. Other metrics could have been used as well, such as the number of vulnerabilities in the SDKs (e.g., implementation failures that can be exploited). If the number of vulnerabilities inside the SDKs would have been included to the analysis, the results may only be valid for a short period of time since both SDKs is in an early stage of development, and implementation failures may be mitigated continuously.

Risk assessment

The following list summarizes the possible errors that may have affected the risk assessment’s results, and the impact such errors could have on the results:

- Vulnerabilities could have been overlooked during the literature study. If a vulnerability is missing, then many threats may be missing as well. If a threat is missing that could have been mitigated by a security property that is not currently identified and not supported in the SDKs, it would have affected the result on how appropriate the SDKs are for protecting VNFs.

- Countermeasures may not be sufficient for mitigating a risk. Useless countermeasures would lead to the use of security properties that is not necessary and can affect how appropriate the SDKs are for protecting VNFs.

- The risk responses could have been designed in different ways with a different amount of security properties. There could be more efficient risk responses that require a smaller amount of security properties, which would have enabled more hardware types to be a possible candidate for protecting VNFs.

A risk assessment traditionally contains an estimate on the impact of exercised vulnerabilities for being able to estimate the risks [58] but was not included to the risk assessment in this thesis because the impact would require a specific purpose and context for the VNF. The risk assessment was only used in this thesis to establish what and how current VNF threats can be mitigated, without narrowing the scope to a specific type of VNF. Therefore, the estimated impact of the threats was not necessary.

According to the NIST SP 800-30 [58], the accuracy of a risk assessment can be enhanced by combining multiple approaches, e.g., using both the threat-oriented and vulnerability-oriented approach for identifying risks. Unfortunately, there was not enough time for conducting two risk assessments with different approaches and was therefore added to the list of future work (see Section 7.3).

Risk responses

The TEE hardware and security properties included by Maene et al. [2] were used for mitigating the risks. There is a possibility that there exists other TEE hardware with security properties that could be used for mitigating the threats more efficiently than the TEEs used by Maene et al. [2]. If there existed more effective hardware or security properties for mitigating the risks, it would affect how appropriate the SDKs are for protecting VNFs because it could exist another hardware that is better for securing VNFs than SGX.

Supported security properties

The following list summarizes possible errors that could have affected the results from the literature study:

- Security properties could have been missed in the SDKs, therefore, give the SDKs an invalid estimation on its security.
6.3. Ethical considerations

- Information could have been misinterpreted, and security properties can be claimed to be supported even if they are not supported. However, this risk was mitigated by executing existing examples in the SDKs that utilized the security properties.

Performance comparison

The experiments did not collect all 5000 samples of each packet size before the next packet size was used. Instead, 25 iterations were conducted on all packet sizes where each iteration collected 200 samples of each packet size. By dividing the experiments into iterations, two problems are mitigated:

- Any noise that may be caused by other applications that are running on the same host is most likely distributed over many packet sizes and do not destroy the results for one packet size.
- If the experiments require a significant amount of time to execute, and the experiments crash because of an unexpected error, the results from each packet size will be possible to display compared to only the first packet sizes if no iterations were used.

Source criticism

The majority of sources are conference papers, technical reports, and white papers. One master thesis report [65] was included in the related work because it was very related to this thesis. Since the documentation and code for the SDKs are online sources, all sources (except Lazard et al. [60]) for the literature study about the SDKs are online sources.

6.3 Ethical considerations

The performance results do not have a large ethical or social impact. The security results, on the other hand, increase the privacy of VNF users and can make it safer for criminals to use VNFs for illegal purposes, e.g., the criminal can provision a virtualized firewall (vFW) to protect a website that is selling illegal products. If no lawful interception (see Section 2.3) interface is implemented to the provisioned vFW, communication cannot be intercepted because the enclave’s memory is encrypted and the private key used for establishing TLS channels is only known by the vFW. Therefore, it needs to be addressed how lawful interception can be conducted on VNFs that is using TEE hardware as protection (see Section 7.3).
This thesis successfully addressed how Open Enclave and Asylo can be used for protecting VNFs. The impact on security and performance is also addressed and explained further in the rest of this chapter.

7.1 What is the impact of integrating Open Enclave and Asylo to VNFs from a security perspective?

The provided security by each SDK is dependent on the number of supported security properties and was used as a metric to measure Asylo’s and Open Enclave’s security. Open Enclave and Asylo only support SGX for now, which means that VNFs secured with Open Enclave and Asylo would be vulnerable for side-channel attacks (e.g., foreshadow) and DoS attacks. None of the TEE hardware considered in this thesis supported the required security properties for mitigating all threats against VNFs.

Asylo did not support remote attestation, which is a security property identified (in this thesis) as critical for being able to provide signed certificates to VNFs (see Section 4.1). Neither Open Enclave or Asylo supported functionality for code confidentiality, which implies that other techniques need to be used for securing VNF’s against code theft (e.g., TEEshift [60]).

7.2 What is the impact of integrating Open Enclave and Asylo to VNFs from a performance perspective?

The performance experiments revealed that there is no large difference in performance between Asylo and Open Enclave. However, the experiments exposed a couple of issues with Asylo, which supports the fact that Asylo is still in an early phase of development:

- The RTT measurement’s standard deviation was on many occasions significant compared to Open Enclave
- The RTTs in the same experiment were increasing with time for Asylo but not with Open Enclave

The performance experiments revealed that the performance penalty of performing an ECALL decreases when the execution time within the enclave increases. The latency (see
Equation 6.1) from using Open Enclave and Asylo with a hardware SGX in debug mode, compared to using no SGX, was 99% and 24% respectively for signing large packets (1Mb). For small packet sizes (1b), the latency for Open Enclave and Asylo was 1277% and 1637%.

Isolating the TEE functionality into separate containers had a tremendous impact on performance when small packet sizes were used: 56752% and 38706% respectively. Large packet sizes had a lower performance penalty: 223% with Open Enclave and 279% with Asylo.

7.3 Future work

The following list summarize propositions on future work:

- Address how lawful interception can be integrated into VNFs that is using enclaves for protection.
- Verify that the risk responses proposed in Section 5.2 can be used in practice by implementing the countermeasures in a real VNF.
- Extend the risk assessment with other LCM-operations (e.g., migration) to identify if security properties can be used for protecting more aspects of the VNF life-cycle.
- Test using Asylo with communication over gRPC in a real VNF and investigate whether the performance overhead is acceptable in production scenarios.
- Verify the results from the risk assessment by using another approach, e.g., asset/impact-oriented, or threat-oriented.
Appendix
A Analysis of Isolating TEE Functionality Into Separate Containers

This appendix contains an analysis of how performance, flexibility, and security are affected by isolating TEE functionality into a separate container (e.g., Docker container), instead of integrating the TEE functionality to an existing container.

A.1 Performance

By running multiple VNFCs in parallel, a VNF can easily scale its resources depending on the current traffic load. Therefore, considering the current performance overhead when using TEE functionality, it can be a good idea to isolate the TEE functionality into a separate VNFC to increase performance capabilities. However, isolating the TEE functionality would make the performance overhead larger for executing a single TEE command.

A.2 Flexibility

Isolating the TEE functionality into a separate container has several positive effects on flexibility:

- It can exist many different alternative implementations of the same TEE functionality that can be easily replaced depending on the situation, without requiring any changes to the rest of the VNF
- If possible, the VNFC with the TEE functionality can be reused by other VNFs (both the code and during runtime)

A.3 Security

By isolating the TEE functionality into a separate container, the threat model automatically changes because the TEE functionality is no longer executed within the same application. If the TEE functionality is located in a separate container, then the communication is performed over the network. The VNFC may be located in a third-party cloud environment. If the communication channel to the enclave is not secured, then the system is vulnerable to
several attacks and needs to be addressed with the same security measures that are used for communicating over networks i.e., TLS.

**A.4 Summary**

The main drawback by isolating the TEE functionality into a separate container is the performance penalty (the constant delay of executing a single function), and if it is too large, then it would not be feasible in production scenarios. If the performance overhead is acceptable, then there are a lot of benefits from a performance capability (can be scaled) and flexibility perspective.
Bibliography


[22] Network functions virtualisation (nfv); infrastructure overview. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_NFV-INF001v010101p.pdf

[23] Network functions virtualisation (nfv); management and orchestration. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfvm-man001v010101p.pdf


[27] Network functions virtualisation (nfv); use cases. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf


[51] Asylo: Release 0.3.4. [Online]. Available: https://github.com/google/asylo/releases/tag/v0.3.4


[71] Open enclave; supported datatypes; edger8r tool. [Online]. Available: https://github.com/microsoft/openenclave/blob/master/docs/GettingStartedDocs/Edger8rGettingStarted.md


[73] Open enclave; host memory allocation. [Online]. Available: https://openenclave.io/apidocs/v0.5/enclave_8h_a10b3ff4164db3852c41fa431950bebb3.html

[74] Open enclave; repository used for experiments. [Online]. Available: https://github.com/microsoft/openenclave/tree/v0.5.0


[76] Asylo; repository used for experiments. [Online]. Available: https://github.com/google/asylo/tree/v0.3.4-hotfix


