João Carlos Cristo Reis

Bachelor of Computer Science and Engineering

SGX-Enabled TREDIS

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser: Henrique João Lopes Domingos, Assistant Professor,
NOVA University of Lisbon

Examination Committee

Chair:	Name of the committee chairperson
Rapporteurs:	Name of a raporteur
	Name of another raporteur
Members:	Another member of the committee
	Yet another member of the committee

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

October, 2019

SGX-Enabled TREDIS

This document was created using the (pdf)LaTeX processor, based on the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Lorem ipsum.

Acknowledgements

The acknowledgements. You are free to write this section at your own will. However, usually it starts with the institutional acknowledgements (adviser, institution, grants, workmates, . . .) and then comes the personal acknowledgements (friends, family, . . .).

Abstract

Intel SGX hardware is a trust-computing base for applications to protect themselves from potentially-malicious OSes or hypervisors. In cloud and other outsourced computing environments, many users and applications could benefit from SGX. However, legacy applications are not prepared to work out-of-the-box on SGX.

Previous research work have already addressed library emulated OSes targeted to execute unmodified applications on SGX, but a belief has emerged that such approaches will not be interesting in terms of performance and TCB size, making in practice that application code modifications or reengineering is always an implicit and better prerequisite for adopting SGX-enabled computing environments.

(HELP) this next paragraph enters the abstract? or just conclusions after it's done?

In this thesis we intend to study existent library OSes approaches and conduct an experimental evaluation by adopting a recent solution of a OS library emulation to be ported on top of SGX, as a fully-featured library OS that can eventually be adopted to rapidly deploy unmodified applications, with overheads comparable to applications modified to use "shim" layers. Our targeted evaluation will be conducted in virtualizing the Redis Key-Value Store, redesigning and implementing it as a SGX-enabled Trusted Key-Value Store (TREDIS).

Keywords: Intel SGX, REDIS, Trusted Execution Environment, Data Protection, Privacy, Dependability . . .

Resumo

O Intel SGX é uma base de computação confiável para que aplicações se protejam de SOs ou Hipervisores potencialmente maliciosos. Em *cloud* ou noutros ambientes de computação garantidos por terceiros, muitos utilizadores e aplicações podem beneficiar do SGX. Trabalhos já realizados que abordaram a emulação de bibliotecas de SOs visaram a execução de aplicações não modificadas sobre SGX, mas surgiu uma convicção de que esse tipo de abordagem não será interessante no que toca à performance ou ao tamanho da base de computação confiável fazendo com que, na prática, tanto modificações como a reengenharia do código de aplicações estejam sempre implícitos e sejam um melhor pré-requisito para adotar ambientes de computação com SGX.

(HELP) this next paragraph enters the abstract? or just conclusions after it's done?

Nesta tese pretendemos estudar as abordagens já existentes de bibliotecas de SOs e conduzir uma avaliação experimental adotando a solução recente de uma emulação de uma biblioteca de SOs para ser usada sobre o SGX, como sendo uma biblioteca completamente caracterizada que possa eventualmente ser adotada para implantar aplicações não modificadas de forma rápida, com *overheads* comparáveis a aplicações modificadas para usar camadas *"shim"*. O foco da nossa avaliação consistirá na virtualização da base de dados do tipo Chave-Valor *Redis*, redesenhando e implementando-a como uma base de dados Chave-Valor confiável com permissões SGX (TREDIS).

Palavras-chave: Intel SGX, REDIS, Ambiente de Execução Confiável, Proteção de Dados, Privacidade, Confiabilidade . . .

Contents

List of Figures

List of Tables

Listings

Glossary

aliquam
tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

computer
An electronic device which is capable of receiving information (data) in a particular form and of performing a sequence of operations in accordance with a predetermined but variable set of procedural instructions (program) to produce a result in the form of information or signals.

cras viverra
metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat.

donec nonummy
pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo.

integer sapien
est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus.

lorem ipsum
dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.

maecenas lacinia
nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem.

morbi ac
orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

morbi dolor
nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi.

nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo.

name arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.

nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis.

sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus.

Acronyms

CPU	Central Processing Unit
EK	Endorsment Key-pair
EPC	Enclave Page Cache
HAP	High-Assurance Process
HSM	Hardware Security Modules
IaaS	Infrastructure as a Service
OS	Operating System
PCR	Platform Configuration Registers
SGX	Intel Software Security Guard Extensions
SSL	Secure Socket Layer
TCB	Trusted Computing Base
TCE	Trusted Computing Environment
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
VM	Virtual Machine
VMM	Virtual Machine Manager
XOM	eXecute Only Memory

Symbols

*

Chapter 1

Introduction

1.1 Topic Framework and Motivation

The *novathesis* was originally developed to help MSc and PhD students of the Computer Science and Engineering Department of the Faculty of Sciences and Technology of NOVA University of Lisbon (DI-FCT-NOVA) to write their thesis and dissertations Using LaTeX. These student can easily cope with LaTeX by themselves, and the only need some help in the bootstrap process to make their life easier.

However, as the template spread out among the students from other degrees at FCT-NOVA, the demand for am easier-to-use template as grown. And the template in its current shape aims at answering the expectations of those that, although they are not familiar with programming nor with markup languages, so still feel brave enough to give LaTeX a try and rejoice with the beauty of the texts typeset by this system.

1.2 Objective

It is up to you, the student, to read the FCT and/or NOVA regulations on how to format and submit your MSc or PhD dissertation.

This template is endorsed by the FCT-NOVA and even linked from its web pages, but it is not an official template. This template exists to make your life easier, but in the end of the line you are accountable for both the looks and the contents of the document you submit as your dissertation.

1.3 Expected Contributions

1.4 Document Organization

CHAPTER 2

Related Work

We're currently in a period where we start to depend more and more on allowing remote services to access our data and execute our applications. Cloud computing systems require users to trust them with their data. Therefore these systems need a way to assure data privacy and security, thus gaining users trust.

In this chapter we address existing solutions that are able to protect applications from the OS or hypervisor, regardless the machine where they're running on, thus increasing the level of trust an user can deposit in an execution of an application.

These existing solutions are organized in different sections in the following way: Section 2.1 covers protection against untrusted OSes; Section 2.2 covers TEEs and hardware-enabled approaches; In Section 2.3 we cover, in more detail, hardware-enabled TEE solutions used today. Section 2.4 covers shielded applications and frameworks compatible with Intel-SGX, which is the TEE technology we choose for our approach. Finally, in Section 2.4 we make a critical analysis on the topics previously discussed, while covering their main advantages and disadvantages.

2.1 Protection in untrusted OSes

A lot of applications these days depend on sensitive data to operate. Therefore protecting this data must be taken into account while designing the application. One of the things we have to think about is the size of the TCB, and how to reduce it as much as possible without losing the operability of the system. Typically, the host OS is considered safe and trustworthy, although that is not always the case. A compromised OS can give complete access to sensitive data, if not isolated from the application. That's why this is a major security problem and must be tackled in today's systems.

Approaches like Virtual Ghost, Flicker, MUSHI, SeCage, InkTag, Sego, all grant security by isolating the sensitive data from the untrusted OS either by monitoring the application while it runs, or by enforcing memory isolation by using virtualization.

Virtual Ghost [8] provides application security against untrusted OSes by implementing the idea of ghost memory, which is unaccessible for the OS to read or write, as well as providing trusted services like ghost memory management, key management, and encryption and signing services.

It relies in sandboxing to protect the system from the OS, where a thin layer of abstraction is interposed between the kernel and the hardware. This layer works as a library of functions that the kernel can call directly, without the need of higher priviledges being given to it.

Thus, Virtual Ghost protects the system against a direct threat from the OS, without losing significant levels of performance.

Flicker [23] provides secured isolated execution of sensitive code by relying on hardware facilities, such as AMD and Intel commodity processors, to run certain pieces of code in a confined environment, while reducing the TCB to as few as 250 additional lines of code.

When Flicker starts, none of the software already executing can monitor or interfere with it's execution and all traces of it's execution can be eliminated before non-Flicker execution resumes.

Thus, with a small TCB and a good level of isolation during the execution fase, where no data is leaked nor possible to access while inside the confined environment, the system can achieve reliability and security.

MUSHI [42] is designed to deal mainly with multi-level security systems, and provides isolation to individual guest VMs executing in a cloud infrastructure. MUSHI ensures that VMs are instantiated securely and remain that way throughout their life cycle.

It offers: 1- Trusted Execution, where both the kernel and user image, as well as MUSHI itself, are attested upon a VM start using a TPM, and thus defining a trusted initial state; 2- Isolation, where each VM executing on the same machine runs isolated, as a way to guarantee confidentiality and integrity; 3- User Image Confidentiality, by encrypting the user image with a cryptographic key provided by the user itself.

MUSHI guarantees confidentiality and integrity of a VM even during malicious attacks from both inside and outside the cloud environment. It trusts a TCB relatively small, including only the hardware, hardware virtualization, BIOS and System Management Mode, and can be implemented with quite ease using modern commodity hardware containing SMM memory (SMRAM), necessary for the isolation between the host and VM.

SeCage [22] uses hardware virtualization to protect user-defined secrets from potential threats and malicious OSes, by isolating sensitive code and critical secrets while denying to the hypervisor any possibility of intervention during runtime.

It devides the system in compartments, where secret compartments have all the permissions to access and manipulate the user-defined secrets, and a main compartment responsable for handling the rest of the code.

SeCage is designed to assure confidentiality of user-secrets, adding a small overhead while supporting large-scale software. To achieve this, it ensures: 1- Hybrid analysis of secrets, where static and dynamic analysis are combined to define secret compartments to execute secrets, preventing them from being disclosed during runtime; 2- Hypervisor protection, by using hardware virtualization to isolate each compartment; 3- Separating control and data plane, where minimal hypervisor intervention is required to deal with communications between compartments. The hypervisor is limited to define policies on whether two compartments can communicate (control plane) for as long as they conform to those policies.

InkTag [12] also uses a virtualization-based approach in order to grant applications protection from untrusted OSes.

Unlike SeCage, InkTag admits trust in the hypervisor. The hypervisor is responsible to protect the application code, data, and control flow from the OS, allowing applications to execute in isolation, in high-assurance processes (HAP). Trusted applications can communicate directly with the InkTag hypervisor via hypercalls, as a way to detect OS misbehavior.

It introduces a concept called paraverification, which simplifies the hypervisor by forcing the untrusted OS to participate in its own verification. As a result, the OS notifies the InkTag hypervisor upon any update to be made to the state, which the hypervisor can check for correctness. InkTag also isolates secure from unsafe data through hardware virtualization, and allows each application to specify their own access control policies, managing their data's privacy and integrity through encryption and hashing. Other important aspect of InkTag is recoverability. InkTag hypervisor can protect the integrity of files even if the system crashes, by ensuring consistency between file data and metadata upon a crash.

Like InkTag, Sego [19] is also an hypervisor-based system that gives strong privacy and integrity guarantees to trusted applications. To protect applications from untrusted OSes, Sego removes the trust from the OS, relying only on a trusted hypervisor which is assumed to always execute correctly. It also enforces paraverification, where the OS communicates its intentions to the hypervisor, thus keeping track of its behavior.

Sego is designed to execute trusted code in an HAP. After booting the OS, the hypervisor starts the HAP, in a way that the HAP itself can verify its own initial code and data, similar to a TPM. Once running, the hypervisor ensures that the HAP's registers and trusted address space are isolated from the OS. Everytime the HAP wants to perform a system call, it must inform the hypervisor of its intent, so that the hypervisor can verify the OS's activity. HAPs use a small library called libsego as a way to handle system calls and get Sego services without having to change their code. Each HAP also contains an untrusted trampoline code, that uses to interact with the OS. This protects its control-flow,

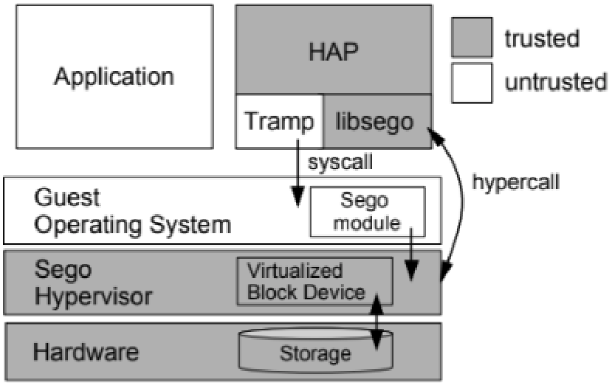

Figure 2.1: Sego Architecture Overview

since it uses this trampoline as issuer for system calls, therefore never compromising the HAP itself. Figure **??** shows well enough Sego's overall design without going into too much dept, indicating which components are included in the TCB and which are not. Context switches are handled by the hypervisor, thus hiding any information about the HAP from the OS.

Sego does not guarantee OS availability. A compromised OS can simply shut down or refuse to schedule processes. However this is easily detected.

————-

drawback of those approaches (inktag, virtghost...) and why aren't enough

———————— END OF TO DO ————————

2.2 Hardware-Enabled TEE - Trusted Execution Environments

A TEE is an abstraction provided by both software and hardware that guarantees isolated execution of specific programs in a machine, including the host OS, hypervisor or even system administrators, preventing them from leveraging their priviledges. A TEE also provides integrity of applications running inside it, along with confidentiality of their assets.

The first attempts to implement a TEE on a cloud system consisted of combining a hypervisor with isolation properties and a TPM.

A TPM [11] consists of a hardware chip, called microcontroller, that aims to create a trustable platform, through encryption and authenticated boot, and make sure it remains trustworthy, through remote attestation. It provides cryptographic functions that can't be modified, and a private key (Endorsment Key) that is unique to every TPM made, working as an identifier for each TPM.

However, TPMs have several problems when applied to the cloud due to being designed with the intention to offer security to a single machine. It is not flexible enough to guarantee that anyone can get the encrypted data from a different node. Thus, a distributed environment would not be the best kind of environment for a TPM to work on.

The current best practice for protecting secrets in a cloud system uses HSMs.

HSM [2] are physical components whose main function is to provide and store cryptographic keys used to encrypt/decrypt data inside a system. HSMs can also perform cryptographic operations (e.g. encryption, hashing, etc.) as well as authenticate through verification of digital signatures and accelerate SSL connections [31], by relieving the servers from some of the workload caused by operations envolving cryptography.

Thus, the system can protect critical secrets, such as cryptographic keys, and support a range of cryptographic functions.

With that in mind, new hardware-enabled solutions were developed to be more flexible and cloud friendly than the TPM, or to incorporate the advantages of HSM to the system. We'll dive into technologies like ARM TrustZone, Intel SGX, AMD-SEV, and others, in the following section.

2.3 Hardware-Enabled TEE Solutions

The idea of using hardware to provide trusted execution environments started appearing as a way to deal with piracy, with examples like TCPA [39] and Microsoft's Palladium being the most known ones. By making use of hardware, it was possible to encrypt data (DVD's, for example) that could only be decrypted by a specific hardware, making it impossible to copy.

Although this approaches were effective back in the day, both of them place their trust in the hardware, not trusting the OS entirely. Thus, since any application does not trust the OS, it does not trust the application to properly use its resources either. Therefore, some of the protection aspects of the OS must be moved into the hardware, as well as changing the interface between the OS and the application so it supports the hardware security features.

XOM, described in the next subsection, was developed as a way to deal with these changes that were required to use this idea of trusted computing in a better way.

2.3.1 XOM

XOM [20], which stands for eXecute Only Memory, is a processor architecture that is able to provide copy protection and tamper-resistence functions, good for enabling code to run in untrusted platforms, deployment of trusted clients in distributed systems like banking transactions, online gaming and electronic voting, but also fundamental to deal with piracy back in the day it was published.

The main idea is to only trust the processor to protect the code and data, thus not trusting the main memory nor any software, including the host OS. However, this idea of only trusting hardware has some implications for OSes design. This happens due to the fact that sharing hardware resources between multiple users is a hard job, specially without trusting any software. It is usually easier to have this policies performed by the OS. Therefore, not trusting the software entirely can sometimes be a drawback.

For XOM architecture to be used, it is required a specific OS (XOMOS). XOMOS runs on hardware that supports tamper-resistant software, and is adapted to manage hardware resources for applications that do not trust it.

XOM offers protection against attackers who may have physical access to the hardware itself, as well as main memory protection, if compromised. For it, the XOM processor encrypts the values in memory and stores the hash of those values in memory as well. It then only accepts encrypted values from memory if followed by a valid respective hash.

2.3.2 ARM TrustZone

ARM Trust Zone [28] are hardware security extentions offered by ARM application processors with the same finality as Intel SGX, create isolated environments where software can execute in a secured and trustable way.

To accomplish this, ARM processors implement two virtual processors backed by hardware access control, where the software stack can switch between two states called secure world (SW) and normal world (NW). The first one has higher priviledges than the second one, therefore it can access NWs copies of registers, but not the other way around. SW is also responsible of protecting running processes in the CPU, while providing secured access to peripherals. Each world acts like a runtime environment and has it's own set of resources. This resources can be partioned between the two worlds or just assigned to one of them, depending on the ARM chip specs.

For the context switch between worlds, ARM processors implement a secured mode called Secure Monitor, where there's a special register responsable of determine if the processor runs code in SW or NW.

Most ARM processors also offer memory curtaining. This consists on the Secure Monitor allocating physical addresses of memory specifically to the SW, making this region of memory unaccessible to the rest of the system.

By default, the system boots always in SW so it can provision the runtime environment before any untrusted code start to run. It eventually transitions to NW where untrusted code can start to be executed.

2.3.3 AMD-SEV

AMD Secure Encrypted Virtualization (SEV) is the AMD approach to provide a TEE, integrated with virtualization. It's a technology focused primarily on cloud computing environments, specifically in public IaaS, as its main goal is to reduce trust from higher priviledged parties (VMMs or OS), so that they can not influence the execution on the other "smaller"parties (VMs).

To achieve this, AMD grants encryption of memory through a technology called Secure Memory Encryption (SEM), or through TransparentSEM (TSEM) if the system runs a legacy OS or hypervisor with no need for any software modifications. After the data is encrypted, SEV integrates it with AMD virtualization architecture to support encrypted VMs. By doing this, every VM is now protected from his own hypervisor (VMM), unabling its access to the decrypted data. Although uncapable of accessing the VM, the VMM is still responsible of controlling each VM's resources. [9]

Thus, AMD provides confidentiality of data by removing trust from the VMM, and creates an isolated environment for the VM to run, where only the VM and the processor can be trusted. However it does not provide integrity of data, allowing replaying attacks to take place, and has a considerably large TCB, since the OS of each VM is trusted. [25]

2.3.4 Sanctum

Just like SGX, the main objective of Sanctum is to offer strong isolation of software modules, altough following a different approach focused in avoiding unnecessary complexity, thus granting a simple security analysis. To make this possible, Sanctum [7], which tipically runs in a RISC-V processor, combines minimal and minimally invasive hardware modifications with a trusted software security monitor that is receptive to analysis and does not perform cryptographic operations using keys.

This minimality idea consists on reusing and slightly modifying existing well-understood mechanisms, while not modifiying CPU building blocks, only adding hardware to the interfaces between blocks, causing Sanctum to be adaptable to other processors in addition to RISC-V.

Sanctum is a pratical approach that shows that a strong software isolation is achievable with a small set of minimally invasive hardware changes, causing reasonably low overhead. This approach provides strong security guarantees dealing with side-channel attacks, such as cache timing and passive address translation attacks.

2.3.5 Intel-SGX

Intel Software Security Guard Extensions (SGX) are a set of instructions built in Intel CPUs, that allow programmers to create TEEs, called enclaves. Enclaves are isolation containers that create an isolated environment where sensitive code can be stored and executed inside, ensuring integrity and confidentiality to it. By doing so, it reduces the TCB in a way that most of the system software, apart from the enclaves and the CPU, is considered not trusted.

Enclaves are mapped into private regions of memory, where only the CPU has access to, reducing the TCB to only the enclave and the CPU itself. Due to this restrictions, not even the most common system libraries can be accessed inside the enclave, since the OS is not considered trusted.

A system that encorporates SGX under it's architecture is divided in two: a trusted component being the enclave, and an untrusted component being the rest of the system. The untrusted one requests the launch of the enclave, where the CPU then manages to allocate the enclave in a private region of memory, made available only to that particular enclave. This portion of memory is kept encrypted in volatile memory, being only decrypted by the CPU if the responsible enclave requests it [4].

Altough isolation is the main objective of SGX, it still allows a way for both untrusted and trusted parties to communicate. This is made possible by the functions ECALL and OCALL. ECALLs are used for an untrusted component to call for trusted code in a secured way - the enclave copies the pointers to that specific code into a buffer, which is then made visible for the untrusted component, ensuring that the untrusted party can't know the real memory address inside the enclave. To communicate the other way arround the enclave calls for an OCALL, where the enclave is temporarily exited, executing then the unstrusted function needed. After that, the enclave is re-entered. OCALLS are mainly used by the enclave to access the network or to deal with I/O disk access.

2.4 SGX-Enabled Frameworks and Shielded Applications

The need for cloud computing is constantly growing in modern applications. It is a cost-effective and pratical solution to run large distributed applications, however the fact that it requires users to trust the cloud provider with their code and data creates some trust concerns for developers.

Although the usage of TEEs aim to tackle this problem by running and storing sensitive data on a isolated environment, protecting that data from unauthorized access. To accomplish this, the main approach is to devide the application into trusted and untrusted parts, reducing the TCB as much as possible as a way to reduce security breaches.

In the next sections we'll discuss frameworks that can accomplish solutions to the problem described above, by working with Intel-SGX as the TEE provider, as a way to complement its regular execution.

2.4.1 Shielded protected applications in untrusted Clouds

As said previously, cloud computing is becoming more and more adopted in today's systems. Therefore, by being a such popular technology, it is a must that their users data remains confidential. However, today's cloud systems are build using a classical hierarchical security model that aims to protect only the cloud provider's code from untrusted code (user's virtual machine), while doing nothing in terms of protecting the users data. Hereupon, the users of a cloud platform must trust the provider's sofware entirely, as well as the provider's staff (i.e. system administrators or anyone with physical access to the hardware).

Several approaches were developed as a way to deal with this potential problem, by implementing the notion of shielded execution for applications running in the cloud.

This concept consists on running server applications in the cloud inside of an isolated compartment. The cloud provider is limited to offering only raw resources (computing power, storage and networking) to the compartment, without being able to access any of the users data, except the one being transmitted over the network.

Assuring a shielded execution of an application fundamentally means that both confidentiality and integrity of the data and code running are granted, and that if the application executes, it behaves just as it is expected. As for the provider, it retains control of the resources, and may protect itself from a malicious guest [3].

2.4.2 SCONE

Container-based virtualization has become quite popular for offering better performance properties than the use of VMs, however it offers weaker isolation guarantees, therefore less security. That's why we observe that containers usually execute network services (i.e. Redis). These are systems that don't need as much system calls as the other services, since they can do a lot via networking, thus keeping a small TCB for increased security.

SCONE [1] is a mechanism for Linux containers that increases the confidentiality and integrity of services running inside them by making use of Intel-SGX.

SCONE increases the security of the system while trying to keep the performance levels reasonable. It does it by: (1) keeping the size of the container's TCB as small as possible, by linking a (small) library inside the enclave to a standard C library interface exposed to container processes. The system calls are executed outside the enclave, and networking is protected by TLS. (2) maximizes the time threads spend inside the enclave by supporting user-level threading and assynchronous system calls, thus allowing a thread outside the enclave to execute system calls without the need for enclave threads to exit. This increases the performance since major performance losses are caused by enclave threads entering/exiting, due to the costs of encrypting/decrypting the data.

2.4.3 Haven

Haven is the first system to achieve shielded execution of unmodified legacy applications for a commodity OS (Windows) and hardware, achieving mutual distrust with the entire host software stack.

It leverages Intel-SGX to protect against privileged code and physical attacks, but also against the challenge of executing unmodified legacy binaries while protecting them from an untrusted host.

Instead of shielding only specific parts of applications and data by placing them inside enclaves, Haven aims to protect entire unmodified applications, written without any knowledge of SGX.

However, there may be problems with this approach. Executing entire chunks of legacy binary code inside an SGX enclave pushes the limits of SGX, and while the code to be protected was written assuming that the OS executing the code would run it properly, this may not be the case. The OS may be malicious. For this latest problem, the so called "Iago attack", Haven uses a library OS adapted from Drawbridge [27], running inside an SGX enclave.

By combining with a remote attestation mechanism, Haven is able to guarantee to the user end-to-end security without the need of trusting the provider.

Although this approach may need a substantial TCB size (LibOS quite large), all this code is inside the enclave, which makes it under user's control.

That's the main goal of Haven: give the user trust by granting confidentiality and integrity of their data when moving an application from a private area to a public cloud.

2.4.4 OpenSGX

OpenSGX [16] was developed as a way to help with the access to TEE software technologies, since these type of technologies were only available for a selected group of researchers. It was made available as an open source platform, and by providing TEE and OS emulation, it contributed a lot for expanding the possibility of research in this area, as well as promoting the development of SGX applications.

OpenSGX emulates the hardware components of Intel-SGX and it's ecosystem, including OS interfaces and user library, as a way to run enclave programs. To emulate Intel-SGX at intruction-level, OpenSGX extended an open-source emulator, QEMU. It's pratical properties result due to six components working with each other: (1) Hardware emulation module: SGX emulation, by providing SGX instructions, data structures, EPC and it's access protection, and SGX processor key; (2) OS emulation: since some SGX instructions are priviledged (should be executed by the kernel), OpenSGX defines new system calls to perform SGX operations, such as dynamic memory allocation and enclave provisioning; (3) Enclave loader: enclave must be properly loaded to EPC; (4) User library: provides a library (sgxlib) with a useful set of functions to be used inside and outside the enclave; (5) Debugging support; (6) Performance monitoring: allow users to collect performance statistics about enclave programs.

In Figure **??** we see an overall example of this framework, where a regular program (Wrapper) and a secured program (Enclave Program) both run as a single process in the same virtual address space. Due to the fact that Intel-SGX uses privilege instructions to setup enclaves, the requests from the Wrapper programare handled by the OpenSGX set of system calls.

OpenSGX was proven capable of running nontrivial applications and promotes the implementation and evaluation of new ideas. By being the first open-source framework to emulate a SGX environment, it was shown to be fundamental to the growth in the TEE area.

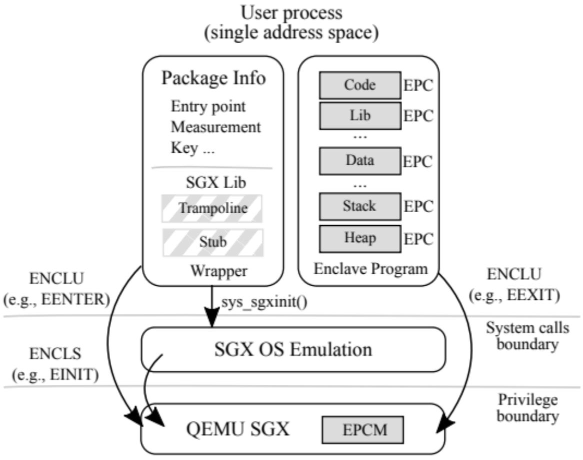

Figure 2.2: Overview design of OpenSGX framework and memory state of an active enclave program

2.4.5 Panoply

Panoply [35] is a system that works as a bridge between SGX-native abstractions and standard OS abstractions, required by most commodity Linux applications. It divides a system into multiple components, called micro-containers (or "micron"), and runs each one of them inside its own enclave. However, when microns communicate with each other, their communication goes through a channel under the OS control. Thus, Panoply design goals are focused on supporting OS abstractions with a low TCB, while also securing inter-enclave communications.

Panoply's design consists of a set of runtime libraries (shim library included) and a build toolchain that helps developers to prepare microns. With this, a programmer can assign annotations to functions to specify which micron should execute a specific function. If not assigned any annotation, a function will be designated to a default micron, who shall execute it. Inter-micron flow integrity is also provided during this stage.

Upon a micron start, Panoply gives it a micron-id which will be used for all further interactions with other microns. It will use this id as a way to assure that only authorized microns can send/receive messages. To extend this inter-micron interaction security, Panoply also provides authenticated encryption of every message, makes use of unique sequence numbers, and acknoledgement messages are sent for every inter-enclave communication. Thus, Panoply can protect microns from silent aborts, replaying, or tampering attacks.

Unlike many other SGX-based frameworks, Panoply supports unlimited multi-threading and forking. Multi-threading in a way that, if a micron reaches its maximum concurrent thread limit, a new micron is launched and all shared memory operations are safely performed. Forking is achieved by replicating a parent micron's data and sending it to the child, over a secure communication.

2.4.6 VC3

Verifiable Confidential Cloud Computing (VC3) [32] is a framework that achieves confidentiality and integrity of data, as well as verifiability of execution of code with good performance through MapReduce [10] techniques. It uses Intel SGX processors as a building block and runs on unmodified Hadoop [36]. In VC3 users implement MapReduce jobs, compile and encrypt them, thus obtaining a private enclave code E-. They then join it with a small portion of public code E+, that implements the protocols for key exchange and job execution. Users then upload the resulting binary code to the cloud, where enclaves containing both E- and E+ are initialized by an untrusted framework F.

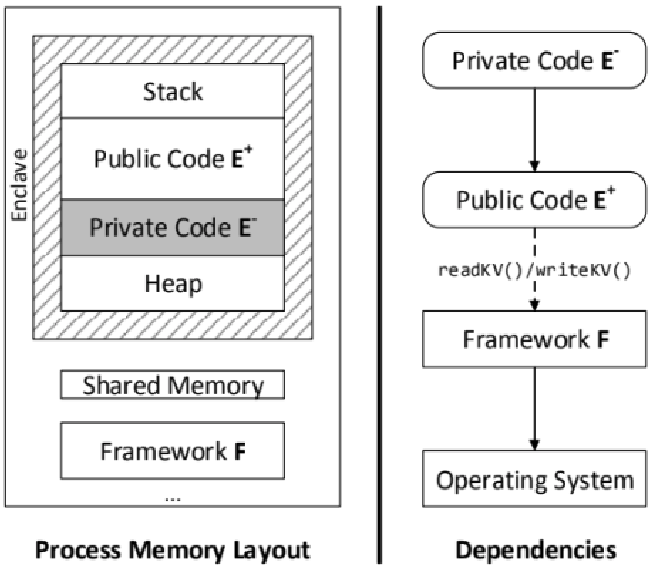

Figure 2.3: VC3 memory design model on the left, component dependencies on the right

A MapReduce begins with a key exchange between the user and the E+ code running in the enclave. After this, E+ can proceed to decrypt E- and process the encrypted data. VC3 isolates this processing from the OS by keeping an interface between the E+ layer and the outside of the enclave. This interface consists of basically two functions: readKV() and writeKV(), for reading a key-value pair on Hadoop or write it, respectively. Also, the data inside of the enclave is passed to the outside, more specifically from E+ to the untrusted F, by using a virtual address space shared by both. With VC3, both E- and the user data are always encrypted while in the cloud, except when processed by the trusted processor, while allowing Hadoop to manage the execution of VC3 jobs. Map and reduce nodes are seen as regular worker node to Hadoop, therefore Hadoop can provide its normal scheduling and fault-tolerance mechanisms, as well as load balancing. VC3 accomplishes this keeping Hadoop, the OS and the hypervisor out of the TCB.

2.4.7 Protected Zookeeper

ZooKeeper [13] is a replicated synchronization service for distributed systems with eventual consistency. However, ZooKeeper does not guarantee privacy of data stored inside of it by default.

Protected ZooKeeper [6] is an approach that eliminates this privacy concerns, by placing an aditional layer between the client and the ZooKeeper, refered as ZooKeeper Privacy Proxy (ZPP). ZPP is a layer responsible for the encryption of all sensitive information, during a communication between a client and the ZooKeeper. Clients communicate with the proxy via a SSL connection, where the packets are encrypted by an individual session key. Here, ZPP acts like a normal ZooKeeper replica to the client. After receiving the packets from the client, ZPP extracts the sensitive data, encrypts it with a mechanism that allows the data to be decrypted by the proxy later on, and forwards the encrypted packet to a ZooKeeper replica where it can be stored with integrity ensured.

ZPP runs inside a TEE, located in the cloud, allowing it to store encryption keys and process plaintext data safely. As a result, even if the attacker is the cloud provider itself, the integrity of the data will still be granted since the attacker won't be able to access or alter anything running inside the TEE.

ZPP also retains all original ZooKeeper functionality, and does not affect ZooKeeper's internal behaviour. Therefore adapting existing ZooKeeper applications to this concept of ZPP it's easily done.

This approach allows applications in the cloud to use ZooKeeper without privacy concerns at the cost of a small decrease of throughput.

2.4.8 Ryoan

Ryoan [14] consists on a distributed sandbox that allows users to protect the execution of their data. This is achieved with the help of Intel SGX [15] [24] enclaves, creating sandbox instances that protect data from untrusted software while also preventing leaks of data, which is a weakness of enclaves caused by side channel attacks. Ryoan does not include any priviledged software (e.g. OS and hypervisor) in it's TCB, while trusting only the hardware (SGX enclave) to assure secrecy and integrity of the data.

It's main goal is to prevent leakage of secret data. This is done by preventing modules from sending sensitive data over their communications if outside the system boundaries, as well as eliminating stores to unprotected memory and system calls, made possible by the use of a trusted sandbox Native Client (NaCl).

Ryoan's approach consists on confining the untrusted application in a NaCl, responsable of controlling system calls, I/O channels and data sizes. This NaCl sandbox is implemented inside the enclave, and can communicate with other instances of the NaCl, forming a distributed sandbox between users and different service providers. Inside the sandbox, the untrusted application can execute safely on secret data. The NaCl sandbox

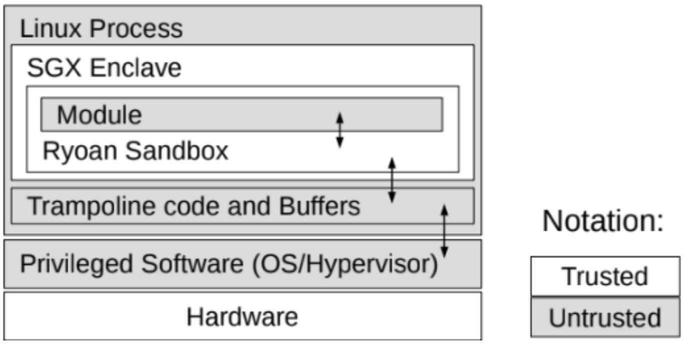

Figure 2.4: Instance of Ryoan running on a single machine

uses a load time code to ensure that the module can not do anything it shouldn't, thus violating the sandbox. To handle faults, exceptions or errors inside the NaCl sandbox, Ryoan uses an unprotected trampoline code, that can enter the enclave and read the information about the fault so it can handle it.

2.4.9 Opaque

Opaque [43] is a distributed data analytics platform that guarantee encryption, secure computation and integrity to a wide range of queries. Therefore, instead of being implemented in the application layer or the execution layer as this kind of security approaches usually are, Opaque is implemented in the query optimization layer.

It is implemented with minimal modifications on Spark SQL, and uses Intel SGX technology as a way to grant confidentiality and integrity of the data. However, the use of enclaves can still be threatened by access pattern leakage that can occur at memory-level, when a malicious OS infers information about encrypted data just by monitoring memory page accesses, and at network-level, when network traffic reveals information about encrypted data.

Opaque hides access patterns in the system by using distributed oblivious relational operators and optimizes these by implementing new query planning techniques. It can be executed in three modes

- encryption mode: provides data encryption and authentication, while granting correct execution.
- oblivious mode: provides oblivious execution, protecting against access pattern leakage.
- oblivious pad mode: extends the oblivious mode by adding prevention of size leakage.

Opaque is an approach that is able to grant oblivious execution 3 times faster than other specialized oblivious protocols.

2.4.10 Graphene-SGX

The usage of Intel SGX, and similar technologies, have proven to add a great sense of privacy to the storage and execution of data in applications. However this tecnologies impose restrictions (e.g., disallowing system calls inside the enclave) that require the applications to be changed so they can benefit from this extra layer of security.

Graphene-SGX [40] came to help circunvent those restrictions, while still assure security to the data. It is a library OS that aims to reproduce system calls, respecting security concerns, so that unmodified applications can use them to keep executing normally without interacting directly with the OS or hypervisor.

By using a library OS, the system is expected to lose performance and, since a new layer of software was added, to increase the size of the TCB. Although these assumptions are true, they are quite often exagerated. Graphene-SGX's performance goes from matching a Linux process to less than 2x in most executions of single-processes. Graphene-SGX has also shown some great results comparing it to other similar approaches that use shim layers, such as SCONE [1] and Panoply [35], where it shows to be performancewise similar to SCONE and faster 5-10 percent than Panoply, while adding 54k lines of code to the TCB compairing to SCONE's 97k and Panoply 20k.

Graphene's main goal is to run unmodified applications on SGX quickly. Thus, whilst the size of the TCB is not the smallest compairing to the other approaches, developers can reduce the TCB as needed as a way to reach a more optimal solution.

Graphene-SGX also supports application partitioning, enabling it to run small pieces of one application in multiple enclaves. This can be useful, for instance, to applications with different privilege levels while still increasing the security of the application.

2.4.11 Other approaches

——————— TO DO ———————

Need any intro or just go straight to the subsections?

——————— END OF TO DO ———————

2.4.11.1 SGX-Enabled Network Protocols and Services

The increased need for security seen nowadays caused network related technologies to also become popular, from security protocols (TLS) to anonymous browsers (Tor), leading to a lot of effort by the community to make viable approaches to tackle network security problems. Hardware approaches capable of providing TEEs (e.g. Intel-SGX) have recently been used to make some contributions to deal with modern network security concerns as a way to, for example, solve policy privacy issues in inter-domain routing, thus protecting ISPs policies.

In [18] it's shown that leveraging hardware protection of TEEs can grant benefits, such as simplify the overall design of the application, as well as securely introduce in-network functionality into TLS sessions. The same paper also presents a possible approach to reach security and privacy on a network level, by building a prototype on top of OpenSGX, that shows that SGX-enabled applications have modest performance losses compared to one with no SGX support, while significantly improving it's security and privacy.

Also at the networking security level, the usage of Network Function Virtualization (NFV) architecture by applications nowadays imply the creation of internal state as a way to allow complex cross-packet and cross-flow analysis. These states contain sensitive information, like IP addresses, user details and cached content (e.g. profile pictures), therefore should be a priority to ensure their protection from potential threats. To tackle this vulnerability, S-NFV [34] has proven to be a valid approach, by providing a secure framework for NFV applications, securing NFV states by using Intel-SGX.

S-NFV divides the NFV application in two: S-NFV enclave and S-NFV host. The enclave is responsible to store the states and state processing code, while the host deals with the rest.

In [34] by implementing the S-NFV approach with Snort [30] on top of OpenSGX was concluded that this SGX-enabled approach results in bigger overheads (aprox. 11x for gets and 9x for sets) than an SGX-disabled Snort application, at the cost of extra security.

2.4.11.2 Trusted Cloud-Based System Administration

——————— TO DO ———————

Tese de Mestrado DI-FCT-UNL, Sup. HJ

——————— END OF TO DO ———————

2.4.11.3 SGX-enabled Virtualization

Has we already pointed out previously, Intel-SGX has drawn much attention from the community in the last few years, which also made cloud providers to start implementing SGX into their cloud systems (Microsoft's Azure confidential computing or IBM Cloud are examples of that). Also, new cloud programming frameworks have already been created with the ability to support SGX.

However, while most of the research on Intel-SGX have been concentrated on its security and programmability properties, there are a lot of questions to answer about how the usage of SGX affects the performance of a virtualized system. This is relevant since virtualization is the main building block of cloud computing.

In [26] an exaustive evaluation about the performance of SGX on a virtualized system, concluding that:

(1) Hypervisors don't need to intercept every SGX instruction to enable SGX for being virtualized. Was concluded that there is only one indispensable SGX function, ECREATE, which is responsable to virtualize SGX launch control.

As a result, glssgx on VMs is considered to have an acceptable overhead.

(2) SGX overhead on VMs when running memory-heavy benchmarks consists mainly of address translation when using nested paging. If instead it uses shadow paging, the overhead becomes insignificant. [26] shows that this can be optimized by using shadow paging for EPC to reduce translation overhead, and nested paging for general usage.

(3) On the contrary, when running benchmarks involving many context switches (e.g., HTTP benchmarks), shadow paging performs worse than nested paging.

(4) The use of SGX causes a heavy drop in performance switching between application and enclave, whether it's using virtualization or not. This drop causes server applications using SGX to be affected. [26] specifies that this can be addressed by using mechanisms (e.g. HotCalls [41]) that work as a fast call interface between the aplication and enclave code. This interfaces reduce the overhead of ecalls and ocalls, helping the porting of applications to SGX.

(5) Swapping EPC pages is really expensive, and this also applies for all systems using SGX. Upon the start, SGX measures the contents of the enclave, thus triggering enclave swapping if enclave's memory size is larger than the available EPC size. This was shown in [26] to be optimizable by minimizing enclave's size, thus reducing swapping and consequently increasing enclave performance. Virtualization causes an additional overhead, which increases based on the number of threads running inside the enclave.

Finally, [26] proposes an automatic selection of an appropriate memory virtualization technique, by dynamically detecting the characteristics of a given workload to identify whether it is suitable with nested or shadow paging.

2.4.11.4 SGX-Enabled Linux Containers

Lately, container solutions such as Linux Containers (LXC) and Docker have proved to be compelling alternatives to virtualization in cloud computing systems, due to the fact that they need less computing resources, allowing more deployments per physical machine to take place, as well as reducing infrastructure costs. However, some concerns have been raised since containers share a common OS kernel, therefore any vulnerability of the kernel would put all the other containers on the system in danger.

While various solutions including Haven, Graphene-SGX, SCONE, Panoply have been proposed to secure applications and containers in the cloud environment by using Intel-SGX, these approaches still generate some concerns, like the need to expand the TCB and enclave size, limited support for key features like remote attestation, and ignore hardware constraints on EPC size (instead of relying on EPC page swapping, which lead to serious performance losses). This issues are the result of a still incomplete infrastructure, from the OS to the application layer.

In [38] these exact concerns are addressed by introducing a platform for Linux Containers (LXC) that leverage Intel-SGX in the cloud environment, called lxcsgx. This lxcsgx platform offers an infrastructure that supports: (1) Remote attestation; (2) EPC memory control for containers to prevent malicious overuse of resources; (3) Software TPM that can easily allow legacy applications to use SGX; (4) GCC plugin to assist with the partitioning of applications, thus reducing the TCB.

Lxcsgx was proven in [38] to offer the lowest overhead to the overall system when compared to the previously spoken solutions, while also addressing potential issues these could have.

2.4.11.5 SGX-Enabled Searchable Encryption

——————— TO DO ———————

——————— END OF TO DO ———————

2.4.11.6 ShieldStore

——————— TO DO ———————

https://dl.acm.org/citation.cfm?id=3303951

——————— END OF TO DO ———————

2.4.11.7 EnclaveDB

——————— TO DO ———————

https://ieeexplore.ieee.org/document/8418608

——————— END OF TO DO ———————

2.5 Summary and Discussion

——————— TO DO ———————

ShieldStore vs EnclaveDB

Graphene

Dissertation Approach

——————— END OF TO DO ———————

2.6 Trusted Computing Environments

We're currently in a period where we start to depend more and more on allowing remote services access our data and execute our applications. Cloud computing systems require users to trust them their data. Therefore these systems need a way to assure data privacy and security, thus gaining users trust. That's were Trusted Computing Environments come in handy. A TCE is a concept that came to grant integrity and confidentiality to systems by forcing a certain machine to behave an expected way, while denying any unwanted access to the data while decrypted. This way, even if the system does not run in a trusted machine, it can be expected that it will execute as it should.

TCEs protect the system against components that, in an ideal world, should always be trusted, like the host OS or even system admins, due to the priviledges they've got.

As a result, and since these components can also be malicious, TCEs prevent them from abusing their privileges, thus ensuring the normal execution of the system.

In the following subsections we'll see how to achieve this properties, what hardware technologies can be used to do it and also how each one of them works.

2.6.1 TPM – Trusted Platform Modules

A TPM [11], proposed by the Trusted Computing Group (TCG), is a hardware chip called microcontroller that aims to create a trustable platform and make sure it remains trustworthy.

The microcontroller is identified by an Endorsment Key-pair, which is unique for every TPM and signed by the manufacturer. It also has a Storage Root Key (SRK) that is used to protect other keys and data inside the TPM. This chip is usually found in the motherboard of most machines nowadays, and it's mainly responsible for providing and storing cryptographic keys that can be used by the system to grant data integrity and confidentiality, as well as provide persistent and volatile memory to store these keys [29].

TPMs can also be complemented with software technologies to achieve better results constructing a trusted platform.

As said before, the main objective of a TPM is to create the idea of a trusted platform. This will be provided by three main services: Encryption, Authenticated Boot and Attestation. The first one is used for pretty much every aspect related with security and privacy. The Authenticated boot consists in booting the OS in stages, as a way of keeping track of which code is trustable through the usage of PCR, that store the trusted software hashes. As for the Attestation, we'll see in the next subsection.

2.6.2 TPM – Enabled Software Attestation

TPMs enable the use of Remote attestation, which is the capability of one system to determine if other system can be trusted to run a particular piece of software as expected or not. An example can be seen in Figure 2.5.

This is made possible by having a trusted configuration of state as reference, provided by the PCRs defined in the boot sequence, followed by a remote system that proceeds to challenge the trusted platform (containing the TPM) with a nonce.

Then the platform creates a message with the nonce received previously and the existing configuration and calculates an hash value for that message. With an Attestation Identity Key (AIK), the message is then signed and sent back to the remote system, which then proceeds to decrypt the message with the EK public part, that then compares the result with the hash of the nonce plus the configuration it had at the begining of the challenge.

If the hashes match, the remote system can then identify the TPM platform as a trusted platform.

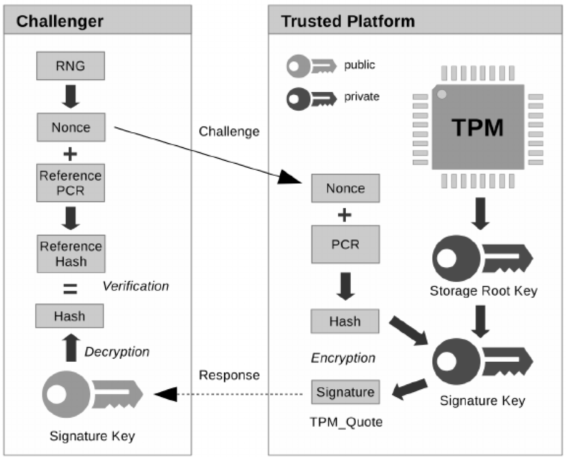

Figure 2.5: Remote Attestation process

2.6.3 HSM – Hardware Security Modules

HSM [2] are physical components whose main function is to provide and store cryptographic keys used to encrypt/decrypt data inside a system. HSMs can also perform cryptographic operations (e.g. encryption, hashing, etc.) as well as authenticate through verification of digital signatures and accelerate SSL connections [31], by relieving the servers from some of the workload caused by operations envolving cryptography.

This modules are used mainly in large environment systems (e.g. large distributed systems) where there are a lot of machines communicating with each other, therefore creating a more needed sense of security. The inclusion of this modules in these big systems is actually a good idea since HSMs can also help servers relieve the workload caused by cryptographic operations. However HSMs do not quite guarantee the idea of absolute security, but increment the cost of attacking the system.

Although HSMs grant some extra level of security to a system, there's some drawbacks. One of them is the cost, where to buy one HSM the price varies depending on the sophistication of the security, plus the cost of maintenance makes it even more expensive. Another disadvantage is the difficulty to update a software module that is already running in a functional HSM.

2.6.4 Trusted Execution Environments

A Trusted Execution Environment is an abstraction provided by both software and hardware that guarantees isolated execution of specific programs from all the other programs running on the same machine [4], including the host OS, hypervisor or even system administrators, preventing them from leveraging their priviledges and thus take advantage of the system. A TEE also grants secured storage of sensitive data, as well as remote attestation to make sure a given program runs as expected on a remote TEE.

For a user to communicate with his program running inside an isolated environment, a key-exchange between the TEE and the user takes place. This way it is ensured both integrity and confidentiality of data during further communications.

This TEE abstraction can be achieved either by using a virtual machine monitor or by running security critical software (from whole applications to little segments of code) under protection mechanisms provided by hardware [5].

In the next chapters we will be looking into more dept about this hardware protection providers, that are capable of creating trusted and isolated environments in the systems nowadays.

2.6.5 Discussion

Although all the approaches discussed before aim to offer better levels of security to applications, it's possible to find some differences between them.

Between the hardware-only approaches TPM and HSM, while TPM is a "fixed"module, since it's typically on the motherboard, HSM is a more flexible approach and can be added to the system after, either as an I/O device or via network. Also HSM is more optimized to be used at the network level, and it's mainly used by banks to perform encryption of user data, while TPM is more suited to deal with encryption of local files.

Adding to this, we also discussed hardware and software approaches. From the four technologies, SGX is the one that proven to be the most reliable for our work. Comparing to ARM Trust Zone, while both really popular in today's systems, SGX supports remote attestation while ARM's approach does not. AMD SEV does not provide protection against replaying attacks, and though it's capable of running whole VMs, it does not support remote attestation, much like ARM's approach. It's also less used by the community than SGX. Sanctum offers a feature that SGX does not offer, which is protection against side-channel attacks. Although the fact that Sactum's implementation was focused on dealing with software attacks resulted in a minimal design for it's enclaves, thus being more susceptible to physical attacks. Also adding the fact that it's the less used approach from all four and it targeted RISC-V processors, made SGX a better option.

Altough SGX does not protect an application from side-channel attacks, it ensures hardware integrity and confidentiality of data, adding the fact that it has proven to be capable of running unmodified applications, by using recently developed frameworks.

2.7 TEE/SGX Enabled Protection Against Untrusted OSes

A lot of applications these days depend on sensitive data to operate therefore protecting these data must be taken into account while designing the application.

One of the things we have to think about is the size of the TCB, and how to reduce it as much as possible without losing the operability of the system. Typically, the host OS is considered safe, trustworthy, although that is not always the case. A compromised OS can give complete access to this sensitive data, regardless of how well designed the application is. That's why this is a major security problem and must be tackled in today's systems.

In the following subsections will be discussed how to achieve security for this particular problem, how to remove the OS from our TCB without losing functionability of the system, preferably without any major drawbacks in the system's performance. Thus we'll introduce some software techniques that can increase the security of an application against that threat.

2.7.1 Virtual Ghost

2.7.2 Flicker

2.7.3 MUSHI

2.7.4 SeCage

2.7.5 InkTag

2.7.6 Sego

2.7.7 Other approaches

There are also other approaches that tackle the same problem of trusting the OS, making it impossible for the execution of an application to be compromised by a malicious OS, such as

- Hardware-assisted Data-Flow Isolation (HDFI) [37] - data isolation mechanism running on top of RISC-V that uses machine instructions and hardware to enforce isolation, by virtually extending each memory unit with an additional tag that is defined by data-flow. It grants stack protection, standard library enhancement, kernel data protection, virtual function table protection, code pointer protection and information leak prevention. It's easy to use and imposes low performance overhead, while improving security.

- Secure Channel between Rich Execution Environment and Trusted Execution Environment (SeCReT) [17] - it is a framework that is focused in securing the communications between the Rich Execution Environments (REE) and the TEE built in ARM TrustZone, to add to the idea of isolation from the OS. It enables legitimate processes to use a session key in the REE, which is regarded as unsafe. To protect the key, SeCReT verifies the code's integrity and control-flow of the process every time a switch between user mode and kernel mode takes place. SeCReT's key-protection mechanism is only activated during the runtime of the process that has permission to access TrustZone, so it minimizes the performance overhead.

2.7.8 Discussion

Another conclusion about the differences of the approaches.

//TODO

Virtual Ghost - performs well compared to previous approaches; it outperforms InkTag on five out of seven of the LMBench microbenchmarks with improvements between 1.3x and 14.3x. For network downloads, Virtual Ghost experiences a 45files and nearly no reduction in bandwidth for large files and web traffic. An application we modified to use ghost memory shows a maximum additional overhead of 5to the Virtual Ghost protections. We also demonstrate Virtual Ghost's efficacy by showing how it defeats sophisticated rootkit attacks.

Protection against untrusted OSes.

Virtual Ghost [20] uses both compile-time and run-time monitoring to protect an application from a potentially- compromised OS, but requires recompilation of the guest OS and application.

Flicker [40], MUSHI [56], SeCage [37], InkTag [21], and Sego [32] protect appli- cations from untrusted OS using SMM mode or virtual- ization to enforce memory isolation between the OS and a trusted application.

Trustlite, isolate software on low-cost embedded devices using a Memory Protec- tion Unit.

Minibox built a 2-way sandbox for x86 by separating the Native Client (NaCl) [55] sandbox into modules for sandboxing and service runtime to support application execution and use Trustvisor [39] to protect the piece of application logic from the untrusted OS.

Secret builds a secure channel to authenticate the appli- cation in the Untrusted area isolated by the ARM Trust- Zone technology.

HDFI extend each memory unit with an additional tag to enforce fine-grained isolation at machine word granularity in the HDFI system.

2.8 SGX-Frameworks and Application Support

2.8.1 Network services protection approaches

Security and privacy have become one of the main concerns for both users and developers, therefore software technologies, like TLS and even anonymous networks like Tor, have become quite popular. At the same time, hardware approaches capable of providing TEEs (e.g. Intel-SGX) have also made contributions to help with this concerns. As these technologies are being adapted by applications, it's also believed that they can have a significant impact on networking security, since they can be used, for instance, to solve policy privacy issues in inter-domain routing, thus protecting ISPs policies.

In [18] it's shown that leveraging hardware protection of TEEs can grant benefits, such as simplify the overall design of the application, as well as securely introduce in-network functionality into TLS sessions. The same paper also presents a possible approach to reach security and privacy on a network level, by building a prototype on top of OpenSGX, that shows that SGX-enabled applications have modest performance losses compared to one with no SGX support, while significantly improving it's security and privacy.

Also at the networking security level, the usage of Network Function Virtualization (NFV) architecture by applications nowadays imply the creation of internal state as a way to allow complex cross-packet and cross-flow analysis. These states contain sensitive information, like IP addresses, user details and cached content (e.g. profile pictures), therefore should be a priority to ensure their protection from potential threats.

To tackle this vulnerability, S-NFV has proven to be a valid approach. S-NFV provides a secure framework for NFV applications, securing NFV states by using Intel-SGX. S-NFV divides the NFV application in two: S-NFV enclave and S-NFV host. The enclave is responsible to store the states and state processing code, while the host deals with the rest.

In [34] by implementing the S-NFV approach with Snort [30] on top of OpenSGX was concluded that this SGX-enabled approach results in bigger overheads (aprox. 11x for gets and 9x for sets) than an SGX-disabled Snort application, at the cost of extra security.

2.8.2 Application-level protection approaches

There are software approaches that allow unmodified applications to execute while offering security from potentially malicious OSes, achieved by isolating sensitive data from the rest.

In addition to Graphene-SGX 2.4.10, approaches like Haven [3], Scone [1] and Panoply [35] offer system support, by ensure a secured way for the application to make system calls, such as implementing a library OS or a standard library, inside the enclave. Haven runs an entire library OS (LibOS) inside the enclave, resulting on a very large TCB. Scone uses sandboxing as a way to reduce TCB size due to the LibOS approach. Panoply provides the abstraction of micro-container (micron), having only to import specific micron-libraries instead of the whole LibOS, resulting in a shorter increase of the TCB size.

Although capable of ensuring unmodified applications a way of running on top of SGX, the increase of the TCB size has been seen as a possible vulnerability. Thus new approaches more focused on this TCB size problem, like Glamdring [21] and SGX-Shield [33], have been developed lately by the community as possible alternatives to the above ones.

2.8.3 Discussion

Haven [15] showed that a library OS could run unmodified applications on SGX,

thin "shim" layers, like SCONE [14] and Panoply [49] wrap an API layer such as the system call table.

SGX frameworks and applications.

VC3 [45] runs MapReduce jobs in SGX enclaves.

Bren- ner et al. [17] run cluster services in ZooKeeper in an enclave, and transparently encrypt data in transit be- tween enclaves.

Ryoan [22] sandboxes a piece of un- trusted code in the enclave to process secret data while preventing the loaded code from leaking secret data.

Opaque [57] uses an SGX-protected layer on the Spark framework to generate oblivious relational operators that hide the access patterns of distributed queries.

The *novathesis* class can be customized with the options listed below.

2.9 Related work analysis and rational

In this section we will provide some additional considerations about some of the customizations available as class options.

CHAPTER 3

Elaboration Work Directions

This Chapter aims at exemplifying how to do common stuff with LaTeX. We also show some stuff which is not that common! ;)

Please, use these examples as a starting point, but you should always consider using the *Big Oracle* (aka, Google, your best friend) to search for additional information or alternative ways for achieving similar results.

3.1 Materialization of objectives and contributions

3.2 System Model and Reference Architecture

3.3 Prototyping effort

3.4 Prototype Validation and Experimental Assessment

3.5 Open issues

Chapter 4

Elaboration Plan

This is the Chapter Four ERROR: File ’chapter5’ does not exist!!!

Bibliography

[1] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O. Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer. "SCONE: Secure Linux Containers with Intel SGX." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 689–703. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov.

[2] J. Attridge. *An Overview of Hardware Security Modules.* 2002. URL: https://www.sans.org/reading-room/whitepapers/vpns/overview-hardware-security-modules-757.

[3] A. Baumann, M. Peinado, and G. Hunt. "Shielding Applications from an Untrusted Cloud with Haven." In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 267–283. ISBN: 978-1-931971-16-4. URL: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/baumann.

[4] G. R. Borges. "Practical Isolated Searchable Encryption in a Trusted Computing Environment." Master's thesis. FCT-UNL, 2018.

[5] B. M. C. Braga. "P-Cop: Securing PaaS Against Cloud Administration Threats." Master's thesis. Instituto Superior Técnico, Universidade de Lisboa, 2016.

[6] S. Brenner, C. Wulf, and R. Kapitza. "Running ZooKeeper Coordination Services in Untrusted Clouds." In: *10th Workshop on Hot Topics in System Dependability (HotDep 14)*. Broomfield, CO: USENIX Association, 2014. URL: https://www.usenix.org/conference/hotdep14/workshop-program/presentation/brenner.

[7] V. Costan, I. Lebedev, and S. Devadas. "Sanctum: Minimal Hardware Extensions for Strong Software Isolation." In: *25th USENIX Security Symposium (USENIX Security 16), year =*.

[8] J. Criswell, N. Dautenhahn, and V. Adve. "Virtual Ghost: Protecting Applications from Hostile Operating Systems." In: *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS 14*. ACM Press, 2014. DOI: 10.1145/2541940.2541986. URL: https://doi.org/10.1145/2541940.2541986.

[9] T. W. David Kaplan Jeremy Powell. *AMD MEMORY ENCRYPTION*. 2016. URL: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf.

[10] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: vol. 51. Jan. 2004, pp. 137–150. DOI: 10.1145/1327452.1327492.

[11] T. C. Group. *Trusted Platform Module (TPM) Summary*. URL: https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Summary_04292008.pdf.

[12] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel. "InkTag." In: *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems - ASPLOS 13*. ACM Press, 2013. DOI: 10.1145/2451116.2451146. URL: https://doi.org/10.1145/2451116.2451146.

[13] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. "ZooKeeper: Wait-free Coordination for Internet-scale Systems." In: *In USENIX Annual Technical Conference*.

[14] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. "Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 533–549. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/hunt.

[15] *Intel Software Security Guard Extentions*. Accessed: 04-07-2019. URL: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html.

[16] P. Jain, S. Desai, S. Kim, M.-W. Shih, J. Lee, C. Choi, Y. Shin, T. Kim, B. Kang, and D. Han. "OpenSGX: An Open Platform for SGX Research." In: Feb. 2016. DOI: 10.14722/ndss.2016.23011.

[17] J. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang. "SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment." In: *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015. DOI: 10.14722/ndss.2015.23189. URL: https://doi.org/10.14722/ndss.2015.23189.

[18] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han. "A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications." In: *Proceedings of the 14th ACM Workshop on Hot Topics in Networks - HotNets-XIV*. ACM Press, 2015. DOI: 10.1145/2834050.2834100. URL: https://doi.org/10.1145/2834050.2834100.

[19] Y. Kwon, A. M. Dunn, M. Z. Lee, O. S. Hofmann, Y. Xu, and E. Witchel. "Sego." In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS 16*. ACM Press, 2016. DOI: 10.1145/2872362.2872372. URL: https://doi.org/10.1145/2872362.2872372.

[20] D. Lie, C. A. Thekkath, and M. Horowitz. “Implementing an untrusted operating system on trusted hardware.” In: *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP 03*. ACM Press, 2003. DOI: 10.1145/945445.945463. URL: https://doi.org/10.1145/945445.945463.

[21] J. Lind, C. Priebe, D. Muthukumaran, D. O. Keeffe, P.-L. Aublin, F. Kelbert, T. Reiher, D. Goltzsche, D. Eyers, R. Kapitza, C. Fetzer, and P. Pietzuch. “Glamdring: Automatic Application Partitioning for Intel SGX.” In: *2017 USENIX Annual Technical Conference (SENIX ATC 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 285–298. ISBN: 978-1-931971-38-6. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/lind.

[22] Y. Liu, T. Zhou, K. Chen, H. Chen, and Y. Xia. “Thwarting Memory Disclosure with Efficient Hypervisor-enforced Intra-domain Isolation.” In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS 15*. ACM Press, 2015. DOI: 10.1145/2810103.2813690. URL: https://doi.org/10.1145/2810103.2813690.

[23] J. M. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki. “Flicker: an execution infrastructure for TCB minimization.” In: vol. 42. Jan. 2008, pp. 315–328. DOI: 10.1145/1352592.1352625.

[24] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas. “Intel Software Guard Extensions (Intel-SGX) Support for Dynamic Memory Management Inside an Enclave.” In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016 on - HASP 2016*. ACM Press, 2016. DOI: 10.1145/2948618.2954331. URL: https://doi.org/10.1145/2948618.2954331.

[25] S. Mofrad, F. Zhang, S. Lu, and W. Shi. “A comparison study of intel SGX and AMD memory encryption technology.” In: *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy - HASP 18*. ACM Press, 2018. DOI: 10.1145/3214292.3214301. URL: https://doi.org/10.1145/3214292.3214301.

[26] T. D. Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and t. Daniel Hagimont. In: 2019. DOI: 10.1145/3311076. URL: https://doi.org/10.1145/3311076.

[27] D. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. Hunt. “Rethinking the Library OS from the Top Down.” In: vol. 46. Mar. 2011, pp. 291–304. DOI: 10.1145/2248487.1950399.

[28] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon, M. Nystrom, D. Robinson, R. Spiger, S. Thom, and D. Wooten. “fTPM: A Software-Only Implementation of a TPM Chip.” In: *25th*

USENIX Security Symposium (USENIX Security 16). Austin TX: USENIX Association, 2016, pp. 841–856. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/raj.

[29] A. M. B. Ribeiro. "Management of Trusted and Privacy Enhanced Cloud Computing Environments." Master's thesis. FCT-UNL, 2019.

[30] M. Roesch and S. Telecommunications. "Snort - Lightweight Intrusion Detection for Networks." In: 1999, pp. 229–238.

[31] N. Saboonchi. "Hardware Security Module Performance Optimization by Using a "Key Pool"." Master's thesis. KTH Royal Institute of Technology in Stockholm, 2014.

[32] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. "VC3: Trustworthy Data Analytics in the Cloud Using SGX." In: *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015. DOI: 10.1109/sp.2015.10. URL: https://doi.org/10.1109/sp.2015.10.

[33] J. Seo, B. Lee, S. Kim, M.-W. Shih, I. Shin, D. Han, and T. Kim. "SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs." In: Jan. 2017. DOI: 10.14722/ndss.2017.23037.

[34] M.-W. Shih, M. Kumar, T. Kim, and A. Gavrilovska. "S-NFV." In: *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization - SDN-NFV Security '16*. ACM Press, 2016. DOI: 10.1145/2876019.2876032. URL: https://doi.org/10.1145/2876019.2876032.

[35] S. Shinde, D. L. Tien, S. Tople, and P. Saxena. "Panoply: Low-TCB Linux Applications with SGX Enclaves." In: *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017. DOI: 10.14722/ndss.2017.23500. URL: https://doi.org/10.14722/ndss.2017.23500.

[36] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. "The Hadoop Distributed File System." In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2010. DOI: 10.1109/msst.2010.5496972. URL: https://doi.org/10.1109/msst.2010.5496972.

[37] C. Song, H. Moon, M. Alam, I. Yun, B. Lee, T. Kim, W. Lee, and Y. Paek. "HDFI: Hardware-Assisted Data-Flow Isolation." In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2016. DOI: 10.1109/sp.2016.9. URL: https://doi.org/10.1109/sp.2016.9.

[38] D. J. Tian, J. I. Choi, G. Hernandez, P. Traynor, and K. R. B. Butler. "A Practical Intel SGX Setting for Linux Containers in the Cloud." In: *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy - CODASPY 19*. ACM Press, 2019. DOI: 10.1145/3292006.3300030. URL: https://doi.org/10.1145/3292006.3300030.

[39] *Trusted Computing Platform Alliance (TCPA)*. 2002.

[40] C. che Tsai, D. E. Porter, and M. Vij. "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX." In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 645–658. ISBN: 978-1-931971-38-6. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai.

[41] O. Weisse, V. Bertacco, and T. Austin. "Regaining Lost Cycles with HotCalls." In: *Proceedings of the 44th Annual International Symposium on Computer Architecture - ISCA 17*. ACM Press, 2017. DOI: 10.1145/3079856.3080208. URL: https://doi.org/10.1145/3079856.3080208.

[42] N. Zhang, M. Li, W. Lou, and Y. T. Hou. "MUSHI: Toward Multiple Level Security cloud with strong Hardware level Isolation." In: *MILCOM 2012 - 2012 IEEE Military Communications Conference*. IEEE, 2012. DOI: 10.1109/milcom.2012.6415698. URL: https://doi.org/10.1109/milcom.2012.6415698.

[43] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. "Opaque: An Oblivious and Encrypted Distributed Analytics Platform." In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 283–298. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng.

Appendix A

Appendix 1 Lorem Ipsum

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum

pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Appendix B

Appendix 2 Lorem Ipsum

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum

pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Annex I

Annex 1 Lorem Ipsum

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum

pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.