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

January, 2020

SGX-Enabled TREDIS

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.

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.

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

xv

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.

xxii

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

DBMS   Database Management System

EPC   Enclave Page Cache

HAP   High-Assurance Process
HSM   Hardware Security Modules

IaaS   Infrastructure as a Service

OS   Operating System

SGX   Intel Software Security Guard Extensions
SSL   Secure Socket Layer

TCB   Trusted Computing Base
TEE   Trusted Execution Environment
TLS   Transport Layer Security
TPM   Trusted Platform Module

VM   Virtual Machine
VMM   Virtual Machine Manager

XOM   eXecute Only Memory

Symbols

*

INTRODUCTION

We are living a new era where data has more value than ever before.

The adoption of mobile devices as an almost indispensable thing in people's everyday lives led to an absurd amount of information produced each second, by everyone, at a global scale. To deal with this, numerous fields of computer science were born, and new technologies more adapted to deal with huge amounts of data were designed. A major one was Cloud Computing, that appeared as a way to provide computing and storage capabilities to systems like we never saw before, while offering fault tolerance and scalability (as well as many other properties) for a reduced cost to their users.

With the adoption of cloud systems, the data could start to be stored and computed in the cloud provider, making the user free of worries concerning physical resources. Hence the popularity of this technology. Both users and smaller companies could now choose to rely on a cloud provider, who are usually hosted by huge corporations, to run their services by paying a monthly fee, thus reducing overall costs.

Although this is nearly perfect, some security problems emerged, more specifically the ones regarding data privacy and security in the provider itself.

The major concern about this was the fact that, after the data reaches the cloud provider, and even if the data is stored safely (encrypted) in disk, there are still no guarantees that the data is encrypted during its execution, while on volatile memory. To deal with this particular problem, a vast amount of solutions were thought to be effective, either by using some kind of virtualization or containerization, or even by relying on the hardware itself on a special way.

Some offer the possibility to isolate the execution of data from the host operating system, making system calls ineffective and blocking typical permissions these might have over the whole system, while others, particularly the more recent ones regarding Trusted Execution Environments, focused on creating a trusted memory region where

data could execute fully encrypted from the outside. In this subject, we highlighted Intel's approach (Intel Software Security Guard Extensions or Intel-SGX) which presented a way to assure privacy in a specific memory region where no one except the machines processor can enter.

However, this kind of technology that isolates data has been proven to have some performance issues, since it involves a lot of encryption and decryption (usually big performance droppers). That implied that some way to increase this kind of technology's performance was needed. This caused many frameworks to place on top of these existent approaches to be introduced (from which we're gonna be looking only at Intel's TEE technology - Intel-SGX - solutions). From improving the performance of storaging encrypted data, to secured networking, all the way to containerization and virtualization of SGX, all these new approaches can add something useful to a system running on top of Intel-SGX. We will be focusing on library OSes approaches, since in [43] was proven to significantly improve the overall performance of a system running on top of Intel-SGX. Library OSes are libraries that puts back into the equation some system calls, in a way that the host OS still can not take advantage of the system's execution by using them.

With that said, and with the recent studies using library OSes, we find that this is the way to go in order to benefit the most from TEE usage, and thus it is worth to explore and find out how a system running on top of SGX performs, with the support of a library OS.

1.1 Topic Framework and Motivation

As cloud computing is getting more attention than ever, the tendency to move data to these cloud providers emerged as a cheap and efficient way to store and compute it. Data is stored in the provider's system, becoming fully dependent of the provider's system security, as well as depending on the provider's employees good behaviour, since they can act maliciously, or just combining curiosity with too much system permissions, exploiting services vulnerabilities. All this brings insecurity to the data, and raises privacy concerns.

However, there are possible ways which can be used to minimize this concerns and that is what we are going to analyse in this thesis, mostly TEE solutions, detailed in 2.3, which appeared as a way to isolate the execution of sensitive data in secured memory regions. Although eliminating most privacy issues, their usage (by themselves) was proven to be big performance droppers ofthe overall execution of a system. This caused numerous studies to take place with the intention to figure out a way to make the most of TEEs whithout crippling too much the system performance.

One of the approaches found to be in the right path to solving this problem was placing libraryOSes on top of the TEE, which enable some system calls to the TEE in order to make the whole execution of data more fluid and fast.

In [43], a libraryOS called Graphene is placed on top of SGX to run Apache web server, GCC and more. It was proven that a system running Graphene can go from matching a regular Linux process performance to performing almost 2x slower than

them. Considering this, [43] shown that is possible to take advantage of TEE properties whithout adding that much overhead to the overall system performance.

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

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 2.5 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 [24] 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 [45] 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 by 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 [23] uses hardware virtualization to protect user-defined secrets from potential threats and malicious OSes, by isolating sensitive code and critical secrets while denying 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 [13] 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 [21] 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,
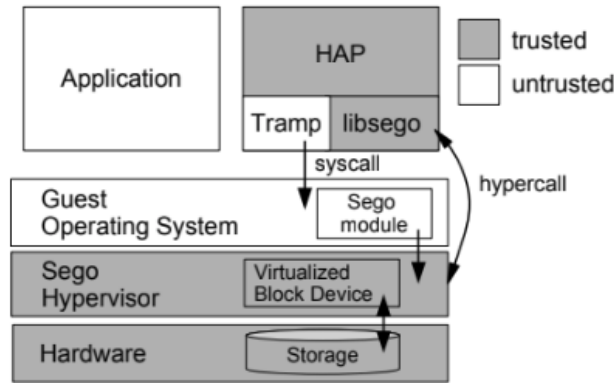
Figure 2.1: Sego Architecture Overview

since it uses this trampoline as issuer for system calls, therefore never compromising the HAP itself. Figure 2.1 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.

In conclusion, although all these approaches are prepared to isolate an application from untrusted hosts, they emphasize the use of software to do it, leaving behind the importance of hardware trustworthiness to the whole system.

In the following sections we will take a look on existing solutions that are able to tackle also hardware related security problems.

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 privileges. 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 [12] 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 the TPM itself.

However, TPMs have several problems when applied to cloud systems, 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 [3] 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 [36], 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 [42] 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 almost 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 [22], 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 TrustZone [32] are ARMs approach to offer a TEE where software can execute in a secured and trustable way, safe from the host machine, as well as its OS and/or hypervisor.

To accomplish this, ARM processors implement two virtual processors backed by hardware access control, where the software stack can switch between two states: 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 its 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 is 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 is 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 (SME), or through TransparentSME (TSME) 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. [27]

2.3.4 Sanctum

The main objective of Sanctum [7] 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, which tipically runs in a RISC-V processor, combines minimal 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 many other processors besides 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 allows programmers to create TEEs, by using enclaves. Enclaves are isolation containers that create a trusted 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. Due to this restrictions, not even the most common system libraries can be accessed inside the enclave, since even the OS is not considered trusted.

A system that encorporates SGX under its 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 [5].

Although 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 around, 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 like SGX aim to tackle this problem by running and storing sensitive data on a isolated environment, protecting that data from unauthorized access, SGX itself has some limitations, and does offer this extra level of security at some costs for the systems.

Hence, to deal with the SGX limitations, some approaches were developed to be implemented on top of it, as a way to make systems more pratical by the integration of trusted computation. We will discuss those approaches in the next subsections.

2.4.1 Shielded protected applications in untrusted Clouds

As said previously, cloud computing is becoming more and more adopted in todays systems. Therefore, by being a such popular technology, it is a must that their users' data remains confidential. However, most of todays cloud systems are build using a classical hierarchical security model that only aims to protect the cloud providers' code from untrusted code (users' virtual machine), while doing nothing in terms of protecting the users' data. Hereupon, the users of a cloud platform must trust the provider software entirely, as well as the provider 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 [4].

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 is 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 [2] 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 host software.

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, executing entire chunks of legacy binary code inside a SGX enclave pushes the limits of the SGX itself, 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, since the OS can be malicious. For this latest problem, the so called "Iago attack", Haven uses a library OS adapted from Drawbridge [30], running inside a SGX enclave. By combining it 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 users 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 [17] 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 its 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. Its pratical properties result of six components working together: (1) Hardware emulation module: SGX emulation, by providing SGX instructions, data structures, EPC and its access protection, as well as 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 2.2 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 program are handled by the OpenSGX set of system calls.

OpenSGX was proven capable of running non-trivial applications, as well as promoting 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 of the TEE field.
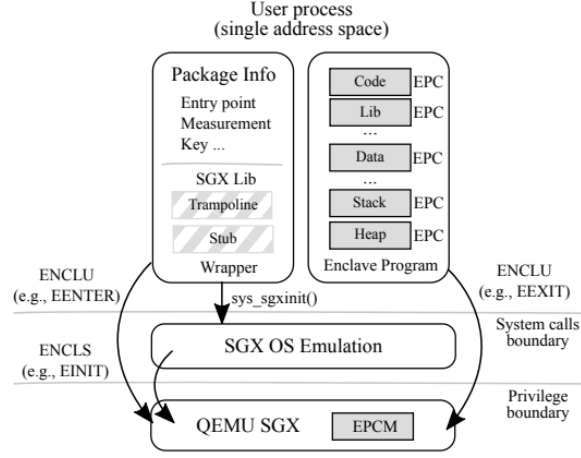
Figure 2.2: Overview design of OpenSGX framework and memory state of an active enclave program

2.4.5 Panoply

Panoply [39] 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 consist on 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 as a way 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.

Each micron is given a micron-id when it starts, 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 protecting the containers 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.

15

2.4.6 VC3

Verifiable Confidential Cloud Computing (VC3) [37] is a framework that achieves confidentiality and integrity of data, as well as verifiability of code execution with good performance through MapReduce [10] techniques. It uses Intel SGX processors as a building block and runs on unmodified Hadoop [40]. In VC3 users implement MapReduce jobs, compile and encrypt them, thus obtaining a private enclave code, named E-. They then join it with a small portion of public code, called 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.
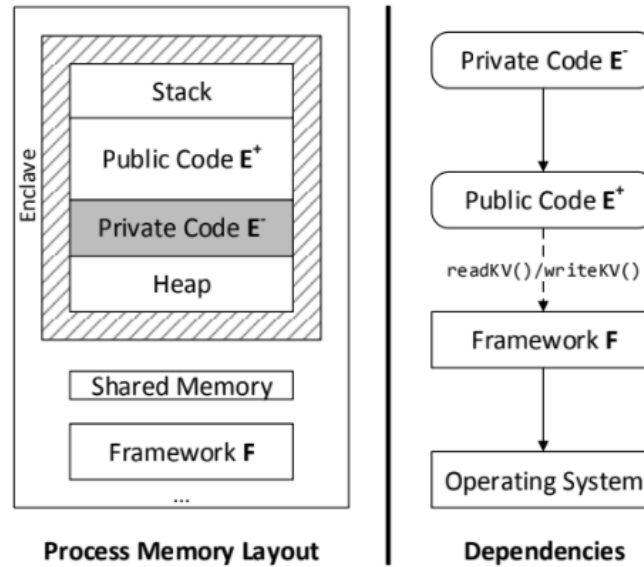

Figure 2.3: VC3 memory design model on the left, component dependencies on the right

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 or writing a key-value pair on Hadoop, respectively. Also, the data inside 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 nodes to Hadoop, therefore Hadoop can keep providing its normal scheduling and fault-tolerance mechanisms, as well as load balancing. VC3 considers Hadoop, as well as both the OS and the hypervisor as untrusted, thus keeping the TCB size as small as possible.

2.4.7 Trusted ZooKeeper Approach

ZooKeeper [14] 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, hence the need to develop an approach capable to add to ZooKeeper this security factor.

Trusted 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 the layer responsible for the encryption of all sensitive information, during a communication between a client and the ZooKeeper. Clients communicate with the proxy via SSL, 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 data safely. As a result, even if the threat comes from 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 can be done with quite ease.

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 [15] consists on a distributed sandbox approach that allows users to protect the execution of their data. This is achieved with the help of Intel-SGX [16] [25] technology, by running NaCl (Google Native Client) sandbox instances inside enclaves, protecting the 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 its TCB. It trusts only the hardware (SGX enclaves) to assure secrecy and integrity of the data.

Its main goal is to prevent leakage of secret data. This is done by keeping the modules from sending sensitive data over communications outside the system boundaries, but also by eliminating the possibility to store data into unprotected memory regions, as well as crossing off the possibility to make most system calls, granted by the use of NaCl instances.

Ryoan's approach consists on confining the untrusted application in a NaCl instance, responsable of controlling system calls, I/O channels and data sizes. This NaCl sandbox is running inside enclaves' memory region, and can communicate with other NaCl instances,
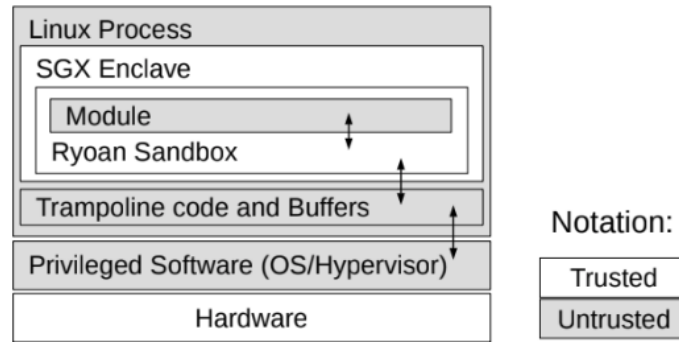
Figure 2.4: Instance of Ryoan running on a single machine

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 uses a load time code to ensure that the module cannot do anything it shouldn't, thus preventing it from 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 [46] 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 Apache Spark [1], a framework for data processing and analytics, and leverages 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 also 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: (1) Encryption mode - provides data encryption and authentication, while granting correct execution. (2) Oblivious mode - provides oblivious execution, protecting against access pattern leakage. (3) Oblivious pad mode - extends the oblivious mode by adding prevention of size leakage.

Opaque is an approach that is able to grant oblivious execution 3x 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 and adapted to this technology, so they can benefit from this extra layer of security.

Graphene-SGX [43] came to help circunvent these restrictions, while still assuring security to the data. It is a libraryOS 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 libraryOS, the system is expected to lose performance and, since a new layer of software was added, increase the size of the TCB. Although these assumptions are true, they are quite often exaggerated. 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 [2] and Panoply [39], where it shows to be performance-wise 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

Other approaches appeared as a way to deal with more specific problems, by making use of trusted computing. We'll go into topics like Networking, Administration of Cloud Systems, Virtualization, Containers, searchable Encryption as well as encrypted Databases, and how to take advantages of SGX in these specific areas.

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 [19] it is 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 it should be a priority to ensure their protection from potential threats. To tackle this vulnerability, S-NFV [38] 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 [38] by implementing the S-NFV approach with Snort [35] on top of OpenSGX was concluded that this SGX-enabled approach results in bigger overheads (aprox. 11x for gets and 9x for sets) than a SGX-disabled Snort application, at the cost of extra security.

2.4.11.2 Trusted Cloud-Based System Administration

The use of cloud platforms lead to a significant increase of security and privacy risks. Thus, cloud services are now higly dependent on trusting its administrators, as well as their good behaviour. Since this is not always the case, it has become a must to protect the users from potencial cloud system administration threats. To tackle this problem, some solutions have been proposed with the help of trusted computing technology. However, their focus have been conducted on Infrastructure-as-a-Service (IaaS) environments, which are simpler to maintain than in PaaS and SaaS approaches.

[34] proposes a solution that addresses trustworthy and security in PaaS and SaaS environments while preserving essential system administration functions by leveraging Intel-SGX. By making use of SGX, this solution provides an environment for cloud customers to review the security conditions of cloud nodes, more specifically those that run their applications and handle their data.

The main idea behind this approach is to allow the administrative staff necessary privileges according to escalation policies enforced for different roles, instead of never granting full administrative privileges on the computing nodes.

The administrative roles for cloud nodes are divided into four or five independent roles, depending if it is a SaaS or PaaS environment, each with their own permissions. Each role works under supervision of internal or external auditors. The internal auditors being the ones hired by the provider, while the external are hired by customers, as a way

to execute protocols decided by both in a trustable way. The solution enables both control of cloud trusted nodes operational states, designated to run customer's computations, as well as remote attestation of the boot sequence of PaaS or SaaS stacks. Finally, by including logging of changes in node's states, this solution is able to offer trustworthy execution of functions and protocols.

This approach was shown in [34] to have minimal performance impact, as well low storage overheads, while being a compelling approach to be used to increase detail of management protocols and tools in cloud environments and data-centers.

2.4.11.3 SGX-enabled Virtualization

As 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 [28] an exaustive evaluation about the performance of SGX on a virtualized system concluded that:

(1) Hypervisors don't need to intercept every SGX instruction to enable SGX to virtualization. It 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 it uses shadow paging instead, then the overhead becomes insignificant. [28] 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 is using virtualization or not. This drop causes server applications using SGX to be affected. [28] specifies that this can be addressed by using mechanisms (e.g. HotCalls [44]) 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 [28] 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, [28] 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 cloud environments 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 leads to serious performance losses). This issues are the result of a still incomplete infrastructure, from the OS to the application layer.

In [41] 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 [41] 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

Processing and storing data in cloud environments is still considered too risky by many. To deal with this, systems started to support TEEs as a way to provide trust to their users, making sure all their data remains private. However, working with encrypted data is not always as easy as it sounds, since software-based approaches made specifically for searching encrypted information still lack some properties, good performance being one of the main ones. Well known approaches like Fully Homomorphic Encryption (FHE), though they offer a good extra security properties, are not pratical in large distributed

systems. As for hardware-based existing approaches, they proved not to scale well due to hardware limitations, as well as depending in a large TCB, which can become susceptible to threats.

CryptDB [29] is a database system capable of processing SQL queries over encrypted data. It is an approach that supports order-preserving encryption for efficient search, which lead to a low performance overhead due to the use of internal index structures, and also allowing range queries on ciphertexts to happen the same way as if the data was in plaintext. However, some research can be seen in [18] about this particular type of encryption, showing that it is possible to recover the original plaintexts, thus making systems like this incapable of providing enough security guarantees to the data.

As a result, new approaches like HardIDX started to become more popular, for providing security properties by supporting trusted computing techniques, while offering decent levels of performance.

HardIDX [11] is a hardware approach that provides the possibility of searching over encrypted data, leveraging Intel-SGX. It implements only a small core of operations, in particular searches (on a single value or value ranges), in the TEE. This approach uses B+-tree as the structure to organize all data, which is found in many DBMSs.

Unlike previous hardware-based approaches, HardIDX implements a small size TCB and memory footprint in the TEE, thus exposing a small attack surface, as well as granting good performance results while executing complex searches on large chunks of data, all this while being able to scale the system to large indexes, as proved in [11].

2.4.11.6 ShieldStore

The increasing research on trusted computation technologies, in particular TEEs like SGX made possible for cloud users to run their applications safely in potentially malicious systems. One of the most used type of applications in these cloud systems are key-value stores, like Redis [33] and memcached [26]. These type of data stores are used in many systems nowadays due to their architecture offering fast access to data, by maintaining data in main memory, as well as granting durability by writing the data to persistent storage. Due to the importance of these type of systems in todays systems stack, there is a special interest in allowing them to be protected by trusted technology. However, one of the critical limitations of SGX is the size of the EPC, which represents its protected memory region.

To circumvent this memory restriction, a in-memory key-value store designed to leverage SGX, ShieldStore [20], was presented. ShieldStore provides fast execution of queries over large data, by maintaining the majority of the data structures outside enclave memory region, hence contributing to overcome SGX memory limitations.

ShieldStore runs inside the enclave to protect encryption keys, remote attestation, and to perform all the logic necessary to the Key-Value Store execution.

ShieldStore design starts by remote attesting the server-side, verifying SGX support of the processor, the code and other critical memory state of an enclave. Then, using Intel-SGX libraries, the client and the server exchange keys so that a secure channel between both parties is created. The client then sends a request, to which the server deciphers and verifies, and thus accessing the Key-Value Store for the desired data. To note that the client does not access the server's ciphertexts neither needs to know the key that the server used to encrypt the values. The server then decrypts the stored data, and encrypts it again with the session key previously decided when establishing the secure channel with the client, and a reply is sent to the client.

This is an approach capable of minimizing the unnecessary memory encryption overhead of paging, and also eliminate EPC page faults, which are the main factors impacting the performance of these kind of encrypted data storage systems. Also, ally to the optimization of the index structures, that enabled them for fast access and protection of keys and values, ShieldStore is proven to be an efficient, reliable Key-Value store capable of leveraging SGX advantages.

2.4.11.7   EnclaveDB

EnclaveDB [31] is an approach to deal also with the need to protect data, both in the storage process and the processing of that data when accessing it. It offers confidentiality and integrity of queries and data, by working alongside Intel-SGX in order to handle all the sensitive data (queries, tables, indexes) inside enclave memory, thus keeping the data safe in cases where the database administrator is malicious, when the OS or hypervisor is compromised, and even when running the database in an untrusted host.

EnclaveDB's design is divided in two modules: trusted, running inside the enclave, and untrusted. The trusted compartment hosts a query processing engine, a transaction manager, pre-compiled stored procedures and a trusted kernel responsible for sealing and remote attestation. As for the untrusted module, it is responsible to run all the other components of the database system.

This approach fundamentally provides a database system with a SQL interface with security guarantees, while dealing with low overhead. Also, by depending on a smaller TCB than any other conventional database server, the security provided increases considerably, making EnclaveDB a valid approach to work as a secured database system.

2.5   Summary and Discussion

After going through all those approaches, there is some things that can be said while choosing which technology set suits better our objective.

Starting with OS isolation systems discussed in 2.1, although capable of assuring a good sense of privacy from the OS/hypervisor, they do not take into account the potencial hardware vulnerabilities that a system can have while making use of sensitive data. With

that in mind, opting for a TEE approach made more sense for offering a more complete sense of security. While TPMs have proven to have problems adapting to the Cloud, modern TEE technology on the other hand has proven to be the way to go, by combining hardware solutions with specific software.

Going deeper into these modern technologies in section 2.3, we had to choose between Intel-SGX and ARM TrustZone from being the two most popular between the community at this point, and so more work is being developed using this two. Intel and ARM's approach differ mainly from the fact that, while ARM TrustZone is designed to implement and work with the idea of only one secured zone, which lead to being widely adopted in the phone industry, Intel-SGX in the other hand is capable of supporting multiple secured zones (enclaves) for each processor, which is a very important subject in cloud systems since a cloud server needs to be capable of managing the execution of processes and data from different users at once. Also by not requiring additional code to grant hardware attestation, since its design also ensures it, Intel-SGX is able to keep a smaller TCB than ARM TrustZone. With this in mind, we choose to go with Intel-SGX in our pratical approach.

Since TEE technologies can drop significantly the overall systems performance when used by itself, in 2.4 we go into detail on how different known technologies can positively impact the work of SGX. Since our focus is in running an unmodified application in this kind of security environments, we opted for Graphene-SGX [2.4.10]. Graphene has proven to be capable of porting unmodified applications on top of SGX without any huge drop of performance, simply by working as a libraryOS that grants access to some restricted and secured system calls.

25

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

4

ELABORATION PLAN

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

Bibliography

[1] M. Armbrust, A. Ghodsi, M. Zaharia, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, and M. J. Franklin. "Spark SQL." In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD 15*. ACM Press, 2015. DOI: 10.1145/2723372.2742797. URL: https://doi.org/10.1145/2723372.2742797.

[2] 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.

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

[4] 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.

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

[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]   B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A.-R. Sadeghi. In: (Mar. 2017).

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

[13]   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.

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

[15]   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.

[16]   *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.

[17]   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.

[18]   S. Kamara and C. Wright. "Inference Attacks on Property-Preserving Encrypted Databases." In: Oct. 2015, pp. 644–655. DOI: 10.1145/2810103.2813651.

[19]   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.

[20] T. Kim, J. Park, J. Woo, S. Jeon, and J. Huh. "ShieldStore: Shielded In-memory Key-value Storage with SGX." In: *Proceedings of the Fourteenth EuroSys Conference 2019*. EuroSys '19. Dresden, Germany: ACM, 2019, 14:1–14:15. ISBN: 978-1-4503-6281-8. DOI: 10.1145/3302424.3303951. URL: http://doi.acm.org/10.1145/3302424.3303951.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] *Memcached: A high performance, distributed memory object caching system*. Accessed: 2019-10-28. URL: http://memcached.org/.

[27] 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.

[28] 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.

[29] R. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. "CryptDB: Protecting confidentiality with encrypted query processing." In: Jan. 2011, pp. 85–100. DOI: 10.1145/2043556.2043566.

[30] 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.

[31] C. Priebe, K. Vaswani, and M. Costa. "EnclaveDB: A Secure Database Using SGX." In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018. DOI: 10.1109/sp.2018.00025. URL: https://doi.org/10.1109/sp.2018.00025.

[32] 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.

[33] *Redis*. Accessed: 2019-10-28. URL: http://www.redis.io/.

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

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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 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.