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

February, 2020

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

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

Acronyms

CPU	Central Processing Unit
DBMS	Database Management System
EPC	Enclave Page Cache
FDE	Full-Disk Encryption
HAP	High-Assurance Process
HSM	Hardware Security Modules
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
KVS	Key-Value Store
OS	Operating System
RAID	Redundant Array of Independent Disks
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

Chapter 1

Introduction

In this chapter we introduce the context and motivation of this dissertation, as well as its problem statement, followed by the goals and expected contributions. In the end we present the structure of the document.

1.1 Context and Motivation

Data has more value than ever before. The adoption of mobile devices as an almost indispensable thing in people's lives led to an imense amount of information produced each second, by everyone, at a global scale [18][19]. 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, which appeared as a way to remotely provide computing and storage capabilities to systems like we never saw before, as well as fault tolerance and scalability (as well as many other properties) for a reduced cost to their users [56]. Thus, the tendency to move data to these cloud providers emerged as an efficient and convinient way to store and compute it, making users free of worries regarding physical resources in their own machines. Both users and companies could now choose to rely on a cloud provider, who are usually hosted by huge corporations, to run their services. However, the complexity behind cloud systems made them major attack targets [37] and security problems, more specifically the ones regarding privacy and security of personal information, were found to be very concerning [39]. Since data is stored in the provider's system, the information is dependent on the provider's security, as well as the behaviour of its staff, which have physical access to the system and can act maliciously. All this brings insecurity to the data and raises privacy concerns. Also, after the data reached the cloud provider, privacy is not assured during execution phase, even if it stored safely (encrypted) in disk, since to be executed it has

to be fully decrypted in volatile memory. The data is then vulnerable during memory attacks, which lead to many efforts to be put into solving this particular problem.

A few 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 [11][31][57][30][17][28] offer the possibility to isolate the execution of data from the host OS, 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 [43][1][10][22], focused on creating a trusted memory region where data could execute fully encrypted from the outside. However, while offering confiability and integrity to the data, TEEs have been proven to have some performance issues, since they depend a lot on encryption and decryption (usually big performance droppers). Unfortunately, the performance loss really impacted their adoption in modern systems, which lead to figuring out a way to make this kind of technology more viable. More recently, a few different approaches to place on top of TEEs have proven to soften the performance issues, creating the impression that we are in the right path to take the most advantage of TEEs.

1.2 Problem Statement

In this dissertation we will study how unmodified applications can be deployed with ease on top of trusted hardware (TEE), and still offer comparable performance overheads relatively to other applications running without this extra layer of security. To achieve this, we'll look into existent library OS approaches and conduct an evaluation by adopting one of them to be ported on top of trusted hardware (SGX).

1.3 Objective and Expected Contributions

Our main objective with this thesis is to implement a secure system based on Intel-SGX where it is possible to deploy unmodified applications without any major performance drop, by adopting the usage of Graphene-SGX to be ported on top of SGX, much like [54] presented. Keeping that in mind, we will deploy an unmodified image of Redis [44] on top of SGX, providing privacy guarantees and isolation of sensitive data, while evaluating if the usage of Graphene-SGX in the system does indeed have a positive impact in the overall performance level. With that said, we expect our solution to deliver the following contributions:

- **Minimize TCB size by using TEE isolation technology**: by adopting TEE technology into our design, and thus promoting isolation of data, we intend to reduce the TCB size as much as possible, ensuring that sensitive data is managed in a controlled way while exposing it to the least possible number of vulnerabilities;

- **Deploy with ease any unmodified application to a TEE based system**: the usage of a fully-featured library OS named Graphene-SGX is thought to be able of making this possible [54];
- **Assuring privacy to sensitive data with usable levels of performance**: again, the adoption of Graphene-SGX is expected to help deliver a better level of performance, as it has proven to be the case in [54]. By working as a library OS, it offers the possibility to run applications with minimal host requirements, making the application safer from its host OS while still taking advantage of system calls.

We expect to achieve a viable Trusted Key-Value Store, where Redis can indeed offer privacy to data by running on top of a TEE solution, with decent levels of performance.

1.4 Report Organization

The remaining of this thesis is organized as follows:

- **Chapter 2** gives an initial background fundamental to understand the objectives and expected contributions of this dissertation, while also covering related work references;
- **Chapter 3** introduces our system approach, covering the model and architecture, as well as some environment setups and implementation guidelines we considered relevant.
- **Chapter 4** presents the defined workplan for the elaboration phase, along with the planned tasks and respective expected duration.

Chapter 2

Related Work

Cloud computing as emerged as an efficient way for modern systems to deal with modern problems, caused by the growth of internet users worldwide over the years [23], where scalability became a must and the cloud's ability to offer storage and computing power on demand made it so usefull. With that said, users (including companies) started choosing cloud providers as a convinient way to store their data and services, trusting that data privacy would be assured. However that is not always the case in modern cloud providers.

In this chapter we address existing solutions able to grant a better level of privacy to data, by protecting applications from the OS/hypervisor regardless the machine they're running on, increasing the level of trust of users in a remote 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

Virtual Ghost [11] 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 needing higher priviledges. Thus, Virtual Ghost protects the system against a direct threat from the OS, without losing significant levels of performance.

Flicker

Flicker [31] provides secured isolated execution of sensitive code by relying on commodity hardware, such as AMD and Intel 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 its traces 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

MUSHI [57] 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 is capable of offering: (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, 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

SeCage [30] 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**, 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

InkTag [17] 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 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.

Sego

Similar to InkTag, Sego [28] 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.

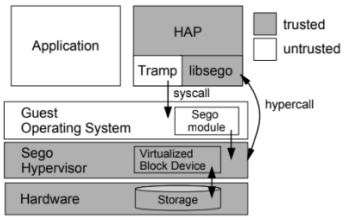

Figure 2.1: Sego Architecture Overview

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, 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 programs in a machine from 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 [16] 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 cloud systems uses HSMs. A HSM [5] is a physical hardware component which provides and stores cryptographic keys used to encrypt/decrypt data inside a system. HSMs also performs cryptographic operations (e.g. encryption, hashing, etc.) as well as authentication through verifying digital signatures and accelerating SSL connections [47], by relieving the servers from some of the workload caused by operations envolving cryptography. Thus, the system can protect critical secrets (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 some other, in the following section.

2.3 Hardware-Enabled TEE Solutions

The idea of using hardware to provide trusted execution environments to run code appeared as a way to deal with piracy, with examples like TCPA [53] and Microsoft's Palladium [34] being the most popular at that time. By providing protection during execution through hardware, it became possible to encrypt data (e.g. DVD's) that could only be decrypted by a specific hardware, making it almost impossible to pirate. 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 should be moved into the hardware, while also changing the interface between the OS and the application so it supports hardware security features.

XOM, described in the next subsection, was one of the first approaches developed as a way to deal with these changes, and one of the stepping stones that lead us to the modern TEE technology we see nowadays, that we will also describe in this section.

2.3.1 XOM

XOM [29] is a processor architecture able to provide copy protection and tamper-resistence functions, usefull 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 [43] is 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 create this abstraction, 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) [1] is the AMD approach to provide a TEE, integrated with virtualization. It is a technology focused 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 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. Thus, AMD provides confidentiality of data by removing trust from the VMM, creating an isolated environment for the VM to run, where only the VM and the processor can be trusted.

However, ADM-SEV 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 [35].

2.3.4 Sanctum

The main purpose of Sanctum [10] 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) [22] 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 [7].

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 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, based on the fact that it is a cost-effective and pratical solution to run large distributed applications. However the fact that it requires users to fully 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 an 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 we said previously, cloud computing is becoming more and more adopted in todays systems. By being such a 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 more worried on the cloud providers' software itself then their users code. Hereupon, for the users of a cloud platform to trust the provider software entirely, as well as the provider staff (i.e. system administrators or anyone with physical access to the hardware), some new measures need to be adapted.

Several approaches were developed as a way to give the user some sense of privacy, by creating 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 offer only raw resources (computing power, storage and networking) to the compartment, without being able to access any of the data, except the one being transmitted over the network. Assuring a shielded execution of an application fundamentally means that both confidentiality and integrity are granted, and that if the application executes, it behaves as it is expected. As for the provider, it retains control of the resources, and may protect itself from a malicious guest [6].

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 [4] 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 keeping the performance levels reasonable. It does it by:

(1) reducing as much as possible the container's TCB, by linking a (small) library inside the enclave to a standard C library interface exposed to container processes. 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 [9], Haven uses a library OS adapted from Drawbridge [41], 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 [24] 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.

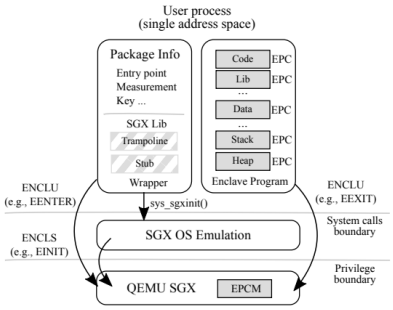

Figure 2.2: Overview design of OpenSGX framework and memory state of an active enclave program

In Figure 2.2 we see an ilustration 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. Since 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, while promoting the implementation and evaluation of new ideas. By being the first open-source framework to emulate a SGX environment, it was fundamental to the growth of the TEE field.

2.4.5 Panoply

Panoply [50] 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.

2.4.6 VC3

Verifiable Confidential Cloud Computing (VC3) [48] is a framework that achieves confidentiality and integrity of data, as well as verifiability of code execution with good performance through MapReduce [13] techniques. It uses Intel SGX processors as a building block and runs on unmodified Hadoop [51]. 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.

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

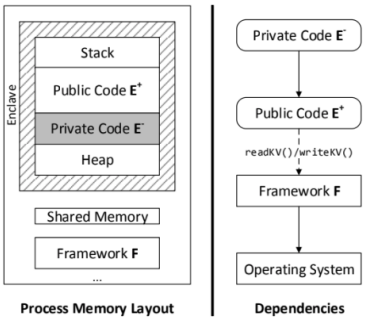

Figure 2.3: VC3 memory design model on the left, component dependencies on the right

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 [20] is a replicated synchronization service for distributed systems with eventual consistency that does not guarantee privacy of stored data by default.

Trusted ZooKeeper was presented in [8] as 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 [21] 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 [22] [32] 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 using 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, 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 [58] 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 [3], 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.

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. However this tecnologies impose restrictions (e.g., disallowing system calls inside the enclave) that require the applications to be adapted to this technology, so they can benefit from their properties.

Graphene-SGX [54] came to help circunvent these restrictions, while still assuring security to the data. It is a libraryOS that aims to reproduce system calls, 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 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 [4] and Panoply [50], 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 SGX-Enabled Networking, Administration of Cloud Systems, Virtualization, Containers, searchable Encryption as well as encrypted Databases, and how to take the best advantages of SGX in these specific areas.

SGX-Enabled Network Protocols and Services

The increased need for security seen nowadays caused network related technologies to 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) are some of those contributions, which 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 [26] 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 adoption 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), creating the necessity to ensure their protection from potential threats. S-NFV [49] 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 [49] by implementing the S-NFV approach with Snort [46] 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.

Trusted Cloud-Based System Administration

The usage 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.

[45] proposes a solution that addresses trustworthiness and security in PaaS and SaaS environments, while preserving essential system administration functions by leveraging Intel-SGX. Thus, 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 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. The approach was shown in [45] to have minimal performance impact, as well low storage overheads, while proving to be a compelling approach that can used to increase the detail of management protocols and tools in cloud environments and data-centers.

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 adopting SGX into their cloud systems (Microsoft's Azure confidential computing or IBM Cloud are examples of that). This lead to the increase of interest in developing new cloud programming frameworks capable of supporting 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, which are considered the main building block of cloud computing.

In [36] an exaustive evaluation about the performance of SGX on a virtualized system made some interesting conclusions:

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, the overhead becomes insignificant. [36] 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) 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. [36] specifies that this can be addressed by using mechanisms (e.g. HotCalls [55]) that work as a fast call interface between the aplication and enclave code, reducing 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 [36] 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, [36] 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.

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, causing any vulnerability of the kernel to be a danger to all the other containers on the system. While various solutions like Haven [2.4.3], Graphene-SGX [2.4.10], SCONE [2.4.2], Panoply [2.4.5] have been proposed to protect applications and containers in cloud environments by leveraging Intel-SGX, these approaches still generate some concerns, since they lead to a growth of the TCB and enclave size, offer limited support for key features (e.g. remote attestation), and ignore hardware constraints on EPC size (instead of relying on EPC page swapping which, on the other hand, leads to serious performance losses). This issues are the result of a still incomplete infrastructure, from the OS all the way to the application layer.

In [52] 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. In the same paper, **Lxcsgx** was proven in [52] to offer the lowest overhead to the overall system when compared to the previously spoken solutions, while also addressing potential issues these could have.

SGX-Enabled Searchable Encryption

Processing and storing data in cloud environments is still not considered thrustworthy enough. Thus, systems started to look at TEEs as a way to change this popular prespective, providing privacy to their users data. 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), although offering 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 on a large TCB, becoming more exposed to threats.

CryptDB [40] is a database system that, although it does not offer isolation to the system (does not support any TEE or isolation technique), we think it is important to mention due to the privacy properties it offers, and also to the contribution it had in this particular field, dealing with security and privacy in data storage. It is capable of processing SQL queries over encrypted data, while supporting order-preserving encryption for efficient search, leading to low performance overheads due to the use of index structures. It also allows range queries on ciphertexts to happen the same way as in plaintext. However, some research [25] was made about this particular type of encryption, proving that it is possible to recover the original plaintexts, which proved to be a big vulnerability, making systems like this incapable of providing the security needed.

As a result, new approaches like Cipherbase [2], and years later HardIDX [15], started to gain relevance for providing security properties to the storage by supporting trusted

computing techniques. Cipherbase appeared as one of the first approaches capable of offering security properties to stored data by leveraging secure hardware and commodity Microsoft servers. It extends Microsoft's SQL Server for supporting efficient execution of queries in a safe way, due to the use of FPGAs [14], which is where we start to see some level of isolation. HardIDX came after, as 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 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, exposing a small attack surface as well as granting good performance results while executing complex searches on large chunks of data. It also offers scalability properties, since it can scale the system list of indexes [15].

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** [44] and **memcached** [33]. 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, it is important to figure out how to protect data inside of these systems by leveraging trusted technology (in our case, Intel-SGX). 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 was designed, ShieldStore [27]. It 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.

Its design starts by remote attesting the server-side, verifying SGX support of the processor, the code and other critical memory state of an enclave. By 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, accessing the Key-Value Store for the desired data. To note that the client does not access the server's ciphertexts or knows the key that the server used to encrypt the values. The server decrypts the stored data, and encrypts it again with the session key previously decided when establishing the secure channel with the client. Finally, a reply is sent to the client. This design minimizes the unnecessary memory encryption overhead of paging, and also eliminates EPC page faults, which are the main factors impacting the performance SGX-enabled systems. Adding to the optimization of the index structures, that enabled them for fast access and protection of keys and values, ShieldStore proved to

be an efficient and reliable Key-Value store capable of taking the best advantages of SGX.

EnclaveDB

EnclaveDB [42] is an approach designed to deal also with protection of data, both when stored and queried. It offers confidentiality and integrity 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, or even when running the database in an untrusted host. EnclaveDB is divided in two modules: trusted, running inside the enclave, and untrusted, outside the enclave. 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 capable of ensuring security guarantees, while dealing with low overheads. 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 trusted database system.

2.5 Summary and Discussion

After going through all those approaches, we can look at the technologies presented in this chapter and pick the ones that we think to suit better our objective for this thesis.

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 knew beforehand that our focus would be in Intel-SGX. However, we picked it from the two most used (along with ARM TrustZone) mainly due to the fact that ARM TZ supports only one secured zone, which lead it to be more adopted by the phone industry, and it is not what we are focusing in this dissertation. On the other hand, Intel-SGX is able to support multiple secured zones (called enclaves) for each processor, which is really relevant in cloud systems due to the need of managing data from multiple users at once. Also by not requiring additional code to grant hardware attestation, since its design supports it, Intel-SGX is able to keep a smaller TCB than ARM TrustZone.

Since the adoption of TEE technologies was found to drop significantly the overall performance of systems when used by itself, in 2.4 we go into detail on how different existent technologies can positively impact the work of SGX. Since our focus resides on

running an unmodified application in this kind of security environments, we opted for Graphene-SGX [2.4.10]. Graphene proved 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.

Chapter 3

Approach of Elaboration Phase

In this chapter we present an initial overview of the system model, as well as some inicial guidelines planned to be implemented to our solution during the elaboration phase, while reinforcing the main objectives of this thesis.

3.1 Addressing the objectives and contributions

The solution we are going to implement and study in this thesis has the purpose of allowing unmodified applications to run in trustable commodity hardware without any major performance overheads, comparing it to applications running without this extra sense of security. To implement this, we are focusing on three fulcral points:

- **Descentralization:** Adding redundancy to the storage and processing of data, increasing the system availability and reliability;
- **Reduction of TCB size:** Making each node of the system only trust what is essencial to its normal execution, leaving the rest out of the TCB will increase the overall security of the system;
- **Data privacy:** Assuring that data remains private while stored and processed, by finding a way to keep the data encrypted, protecting it from entities that are not supposed to access it.

By achieving this, we hope to deliver a full-fledged solution, capable of granting privacy and security to data inside dependable systems.

3.2 System Model Overview

Our system model is composed in three main components: a client, a processing layer made of a KVS that will be responsible of storing and processing data at runtime, and a storage layer. For a better grasp of the system as a whole, we can analyse the Figure 3.1.

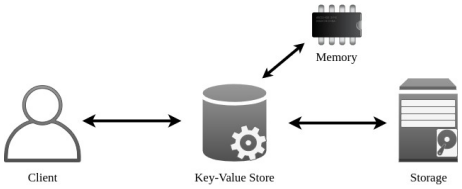

Figure 3.1: Overview of the System Model

The client layer is going to consist of a benchmark application responsable of making requests to the KVS, while evaluating certain defined metrics that will be used on our experimental analysis. The processing layer, as mentioned before, will be accountable of storing and processing the data while executing, making use of volatile memory that should be protected during this phase. As for the storage layer, we will use a RAID based persistent storage system as a way to increase redundancy, thus adding fault-tolerance and enabling faster operations over this third layer.

3.3 Adversary Model

We prevent interventions made by System Administrators on Nodes from having any effect on the computations of users. This means we want to avoid having System Administrators on a node with computations running, however we want to avoid interrupting

the computations or degrading its throughput. If interruptions or degradation happen, we have to make sure these are as small as possible. We assume all nodes in the infrastructure have TPMs in their hardware architectures, and these work correctly, properly providing the functions as we described in the Chapter 2. Only the TPM possesses the proposer TPM's private keys in validated certification chains. Any recipient of the booted configuration (composed by the BIOS configuration, the bootloader, Host OS and all the relevant software components in the Host OS level) can use the TPM's public key and the certification chain (ending in a root of trust) to verify the related signatures. We assume that all nodes in the infrastructure are equipped with Intel SGX enabled

CPUs and the necessary firmware support at the Host OS support level, which is contained in the TCB of the solution.

PaaS software or SaaS applications should be deployed with minimal interaction (e.g. using Preboot eXecution Environment) as stock images and, is assumed correct as far as promised functionality and security are concerned. As we will target on Dockerized images, we consider that Docker does not corrupt applications and is able to contain them, with fully execution isolation. Current cryptographic protocols and standards, as

3.4 Isolation and Containerization

Since our objectives are pointed towards an isolated system capable of offering security and privacy properties, we depend a lot on isolation techniques to make this possible, provided by hardware or software (containers).

As for hardware isolation, we need to look at approaches capable of assuring both computation and storage security to our system's data during runtime. There are several technologies that implement TEEs that were proven to be capable of doing just that. Also, adding to that, by using a container we will grant an extra layer of isolation to the system, as well as ease in the deployment of software running inside the container, whether it is an OS, a library OS or even entire applications.

3.5 System Generalization

Looking at our system model introduced in 3.1, there are some potencial threats that need to be taken care of. First, the connection between the client and the KVS need to be secured. Secondly, the data being stored at the storage layer need to be encrypted, otherwise it can be accessed and read, or even stolen. Lastly, considering only one KVS will induce into a single point of failure on the processing layer. Also, the data stored and running inside this layer and its memory need to be secured.

This can be achieved by using TLS over the HTTP protocol, securing the communication between the client and the KVS layer. Existing FDE technology can be used to encrypt the data to be stored in the storage layer. Finally for the processing layer, we will use a cluster of KVS, thus adding redundancy and load balancing. As for the security and privacy concerns, by isolating each cluster replica inside individual TEEs, we achieve confidentiality and privacy of the data during runtime. The figure 3.2 shows an overall idea of this proposed solutions.

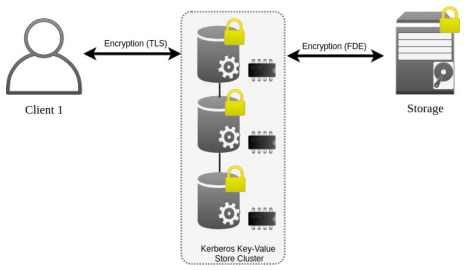

Figure 3.2: System Model with a Key-Value Store Cluster

As a way to scale the overall system to modern requirements, we opted to divide it into two parts, one being the client layer and the other being both the processing and storage layers, which will both be implemented inside a cloud system. With this approach (figure 3.3) we will guarantee scalability of the resources for the solution.

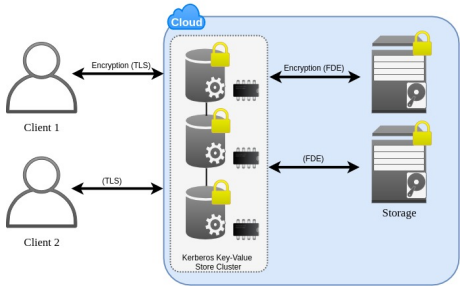

Figure 3.3: System Model running inside a Cloud System

3.6 Implementation Guidelines

For the development phase of this dissertation, and as we mentioned before in 1.3, we considered the following guidelines:

Key-Value Store: we opted for Redis due to being the most popular KVS at the moment, as we can see in [12]. Adding this to the fact that it fits our system needs, we thought that an exhaustive study over this technology running in a dependable system would be an interesting contribution.

Trusted Execution Environment: as mentioned before, we choose to go with Intel-SGX (2.3.5), since it offers the possibility to isolate code inside multiple enclaves, while executing on commodity hardware. Thus, we can run each node of our KVS cluster inside of an individual enclave, assuring confidentiality and privacy of the data during runtime.

OS library: to run unmodified applications on top of Intel-SGX, we picked Graphene-SGX (2.4.10) as it was proven to be able to keep decent levels of performance in a system.

Containerized OS Virtualization: we choose Graphene-SGX Secure Container (GSC) due to being an open-source container system where an application can be protected by Intel-SGX while running inside a container. We thought it to be a good way of adding an extra layer of isolation to the execution of each KVS instance, while still running with the benefits of Graphene-SGX and Intel-SGX.

Cloud Infrastructure: OVH [38] was the cloud provider chosen, mainly due to the fact that it supports SGX in its dedicated servers.

As for other technologies, the system will be written in Java. The client layer will consist of a benchmark application provided by both **Redis Benchmark** and **Yahoo! Cloud Serving Benchmark** (YCSB). Finally, Jedis (Redis java client) will also be used in the processing layer, as the client to our KVS instances.

3.7 Validation and Experimental Analysis

To validate and evaluate our solution, we created a set of tests for each metric we choose to evaluate, comparing it to the regular, already existing, version of the system.

First of all, we will be validating if the implemented solution does indeed result in a

full-fledged solution for a dependable system, if it does guarantee the desired confidentiality and privacy properties to the data.

Then, we will be comparing the following metrics:

- **Attestation latency:** time of the attestation between the client and Redis;
- **Performance:** throughput and latency of the Redis KVS;
- **Resource allocation:** use of resources during runtime.

In order to evaluate these metrics, we will compare scenarios where:

1) a system runs with a regular Redis system model (system model 3.1) and other system implements the TREDIS approach (system model 3.3);

2) the size of the datasets used in the benchmark tools are distinct (e.g: 1,000 entries and 100,000 entries);

3) the typology of operations made over the KVS varies: **(3.1)** ratio of read/write operations (e.g: 20/80, 50/50, 80/20); **(3.2)** time of searches; **(3.3)** time to upload code;

CHAPTER 4

WORKPLAN

Bibliography

[1] *AMD MEMORY ENCRYPTION*. Accessed: 03-07-2019. URL: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf.

[2] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. "Orthogonal Security With Cipherbase." In: *6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*.

[3] M. Armbrust, A. Ghodsi, M. Zaharia, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. Bradley, X. Meng, T. Kaftan, and M. Franklin. "Spark SQL." In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD 15*.

[4] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O Keeffe, M. 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)*.

[5] J. Attridge. *An Overview of Hardware Security Modules*. 2002.

[6] 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)*.

[7] G. Borges. "Practical Isolated Searchable Encryption in a Trusted Computing Environment." Master's thesis. FCT, Universidade Nova de Lisboa, 2018.

[8] 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)*.

[9] S. Checkoway and H. Shacham. "Iago attacks: Why the system call API is a bad untrusted RPC interface." In: *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS* (2013).

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

[11] 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*.

[12] "DB-Engines Ranking of Key-value Stores." In: *DB-Engines* (). Accessed: 16-02-2020. URL: https://db-engines.com/en/ranking/key-value+store.

[13] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Communications of the ACM* (2004).

[14] *Field-Programmable Gate Array*.

[15] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A.-R. Sadeghi. "HardIDX: Practical and Secure Index with SGX." In: 2017.

[16] T. C. Group. *Trusted Platform Module (TPM) Summary*. Accessed: 20-06-2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Summary_04292008.pdf.

[17] O. Hofmann, S. Kim, A. Dunn, M. Lee, and E. Witchel. "InkTag." In: *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems - ASPLOS 13*.

[18] "How much data do we create every day?" In: *Forbes - May '18* (). URL: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#77e3686860ba.

[19] "How much data is created on the internet each day?" In: *Microfocus* (). URL: https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/.

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

[21] 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)*.

[22] *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.

[23] "Internet Growth Statistics." In: *InternetWorldStats* (). Accessed: 13-01-2020. URL: https://www.internetworldstats.com/emarketing.htm.

[24] 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: 2016.

[25] S. Kamara and C. Wright. "Inference Attacks on Property-Preserving Encrypted Databases." In: 2015.

[26] 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*. 2015.

[27] 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*.

[28] Y. Kwon, A. Dunn, M. Lee, O. 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*.

[29] D. Lie, C. 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*.

[30] 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*.

[31] J. M. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki. "Flicker: an execution infrastructure for TCB minimization." In: *EuroSys'08 - Proceedings of the EuroSys 2008 Conference* ().

[32] 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*.

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

[34] *Microsoft Palladium: Next Generation Secure Computing Base*. Accessed: 10-10-2019. URL: https://epic.org/privacy/consumer/microsoft/palladium.html.

[35] 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*.

[36] T. D. Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and t. Hagimont Daniel. In: 2019.

[37] "Over a third of firms have suffered cloud attacks." In: *Infosecurity Magazine* (). Accessed: 20-01-2020. URL: https://www.infosecurity-magazine.com/news/over-third-firms-have-suffered/.

[38] *OVHcloud*. Accessed: 16-02-2020. URL: https://www.ovh.com/world/.

[39] "Playstation Network attack." In: *The Guardian 2011* (). Accessed: 20-01-2020. URL: https://www.theguardian.com/technology/2011/apr/26/playstation-network-hackers-data.

[40] R. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. "CryptDB: Protecting confidentiality with encrypted query processing." In: *SOSP'11 - Proceedings of the 23rd ACM Symposium on Operating Systems Principles* ().

[41] D. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. Hunt. "Rethinking the Library OS from the Top Down." In: *Sigplan Notices - SIGPLAN* (2011).

[42] C. Priebe, K. Vaswani, and M. Costa. "EnclaveDB: A Secure Database Using SGX." In: *2018 IEEE Symposium on Security and Privacy (SP)*.

[43] 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)*.

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

[45] A. Ribeiro. "Management of Trusted and Privacy Enhanced Cloud Computing Environments." Master's thesis. FCT, Universidade Nova de Lisboa, 2019.

[46] M. Roesch and S. Telecommunications. "Snort - Lightweight Intrusion Detection for Networks." In: 1999.

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

[48] 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*.

[49] 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*.

[50] S. Shinde, D. Le Tien, S. Tople, and P. Saxena. "Panoply: Low-TCB Linux Applications with SGX Enclaves." In: *Proceedings 2017 Network and Distributed System Security Symposium*.

[51] 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)*.

[52] D. Tian, J. Choi, G. Hernandez, P. Traynor, and K. 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*.

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

[54] C.-c. Tsai, D. Porter, and M. Vij. "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX." In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*.

[55] 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*.

[56] "Who coined cloud computing?" In: *TechnologyReview* (). Accessed: 20-01-2020. URL: https://www.technologyreview.com/s/425970/who-coined-cloud-computing/.

[57] N. Zhang, M. Li, W. Lou, and Y. Thomas Hou. "MUSHI: Toward Multiple Level Security cloud with strong Hardware level Isolation." In: *MILCOM 2012 - 2012 IEEE Military Communications Conference*.

[58] W. Zheng, A. Dave, J. Beekman, R. Ada Popa, J. Gonzalez, and I. Stoica. "Opaque: An Oblivious and Encrypted Distributed Analytics Platform." In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.