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

September, 2019

SGX-Enabled TREDIS

Copyright © João Carlos Cristo Reis, Faculty of Sciences and Technology, NOVA University Lisbon.
The Faculty of Sciences and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

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

Lorem ipsum.

ACKNOWLEDGEMENTS

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

ABSTRACT

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

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

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

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

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

RESUMO

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

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

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

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

CONTENTS

1 Introduction **1**

1.1 Topic Framework and Motivation . . . 1

1.2 Objective . . . 1

1.3 Expected Contributions . . . 2

1.4 Document Organization . . . 2

2 Related Work **3**

2.1 Trusted Computing Environments . . . 3

2.1.1 TPM – Trusted Platform Modules . . . 4

2.1.2 TPM – Enabled Software Attestation . . . 4

2.1.3 HSM – Hardware Security Modules . . . 5

2.1.4 Trusted Execution Environments . . . 5

2.1.5 Intel SGX . . . 6

2.1.6 RISC-V enabled Sanctum . . . 6

2.1.7 ARM Trust Zone . . . 7

2.1.8 AMD SEV . . . 7

2.1.9 Discussion . . . 8

2.2 TEE/SGX Enabled Protection Against Untrusted OSes . . . 8

2.2.1 Virtual Ghost . . . 9

2.2.2 Flicker . . . 9

2.2.3 MUSHI . . . 10

2.2.4 SeCage . . . 11

2.2.5 InkTag . . . 11

2.2.6 Sego . . . 12

2.2.7 Other approaches . . . 13

2.2.8 Discussion . . . 13

2.3 SGX-Frameworks and Application Support . . . 14

2.3.1 VC3 protection for MapReduce Jobs . . . 15

2.3.2 Protected ZooKeeper . . . 16

2.3.3 Ryoan Sandboxing . . . 16

2.3.4 Opaque . . . 17

2.3.5 Graphene-SGX . . . 18

2.3.6 Network services protection approaches 18
2.3.7 Application-level protection approaches 19
2.3.8 Discussion 19
2.4 Related work analysis and rational 20

3 Elaboration Work Directions **21**
3.1 Materialization of objectives and contributions 21
3.2 System Model and Reference Architecture 21
3.3 Prototyping effort 21
3.4 Prototype Validation and Experimental Assessment 21
3.5 Open issues 21

4 Elaboration Plan **23**

Bibliography **25**

Apêndices **31**

A Appendix 1 Lorem Ipsum **31**

B Appendix 2 Lorem Ipsum **33**

Annexes **35**

I Annex 1 Lorem Ipsum **35**

LIST OF FIGURES

2.1 Remote Attestation process 5
2.2 Sego Architecture Overview 12
2.3 VC3 memory design model on the left, component dependencies on the right 15
2.4 Instance of Ryoan running on a single machine 17

LIST OF TABLES

LISTINGS

GLOSSARY

aliquam	tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.
computer	An electronic device which is capable of receiving information (data) in a particular form and of performing a sequence of operations in accordance with a predetermined but variable set of procedural instructions (program) to produce a result in the form of information or signals.
cras viverra	metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat.
donec nonummy	pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo.
integer sapien	est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus.
lorem ipsum	dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.
maecenas lacinia	nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem.
morbi ac	orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.
morbi dolor	nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

nam lacus	libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi.
nam dui	ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo.
name arcu	libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.
nulla malesuada	porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis.
sed lacinia	nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus.

ACRONYMS

CPU	Central Processing Unit
EK	Endorsment Key-pair
HSM	Hardware Security Modules
IaaS	Infrastructure as a Service
OS	Operating System
PCR	Platform Configuration Registers
SGX	Intel Software Security Guard Extensions
SSL	Secure Socket Layer
TCB	Trusted Computing Base
TCE	Trusted Computing Environment
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
VM	Virtual Machine
VMM	Virtual Machine Manager

SYMBOLS

*

Chapter 1

Introduction

This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-nc/4.0/`*.*

1.1 Topic Framework and Motivation

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

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

1.2 Objective

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

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

1.3 Expected Contributions

1.4 Document Organization

Chapter 2

Related Work

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 (whether locally or remotelly), 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 Trusted Computing Environments; Section 2.2 covers some relevant approaches that enable protection against untrusted OSes by making use of Trusted Execution Environments; Section 2.3 covers Intel-SGX frameworks;

Finally, in Section 2.4 we make a critical analysis on the previously discussed techniques, while covering it's main advantages and disadvantages.

2.1 Trusted Computing Environments

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

TCEs protect the system against components that, in an ideal world, should always be trusted, like the host OS or even system admins, due to the priviledges they've got. As a result, and since these components can also be malicious, TCEs prevent them from abusing their privileges, thus ensuring the normal execution of the system.

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

2.1.1 TPM – Trusted Platform Modules

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

The microcontroller is identified by an Endorsment Key-pair, which is unique for every TPM and signed by the manufacturer. It also has a Storage Root Key (SRK) that is used to protect other keys and data inside the TPM. This chip is usually found in the motherboard of most machines nowadays, and it's mainly responsible for providing and storing cryptographic keys that can be used by the system to grant data integrity and confidentiality, as well as provide persistent and volatile memory to store these keys [26].

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

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

2.1.2 TPM – Enabled Software Attestation

TPMs enable the use of Remote attestation, which is the capability of one system to determine if other system can be trusted to run a particular piece of software as expected or not. An example can be seen in Figure 2.1.

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

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

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

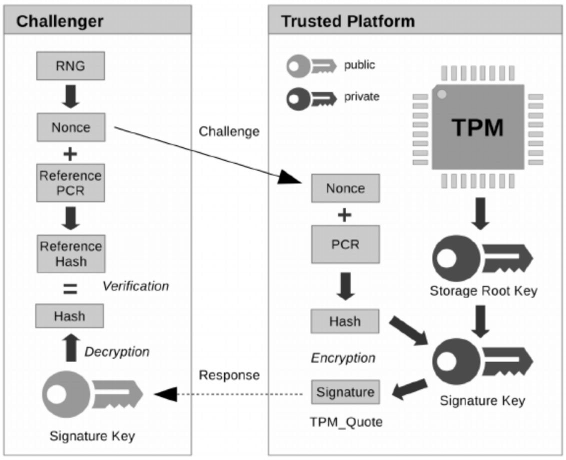
Challenger
RNG
Nonce
+
Reference PCR
Reference Hash
=
Verification
Hash
Decryption
Signature Key
Challenge
Response
Trusted Platform
public
private
TPM
Nonce
+
PCR
Hash
Encryption
Signature
TPM_Quote
Storage Root Key
Signature Key

Figure 2.1: Remote Attestation process

2.1.3 HSM – Hardware Security Modules

HSM [2] are physical components whose main function is to provide and store cryptographic keys used to encrypt/decrypt data inside a system. HSMs can also perform cryptographic operations (e.g. encryption, hashing, etc.) as well as authenticate through verification of digital signatures and accelerate SSL connections [28].

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

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

2.1.4 Trusted Execution Environments

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

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

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

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

2.1.5 Intel SGX

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

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

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

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

2.1.6 RISC-V enabled Sanctum

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

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

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

2.1.7 ARM Trust Zone

ARM Trust Zone [24] are hardware security extentions offered by ARM application processors with the same finality as Intel SGX, create isolated environments where software can execute in a secured and trustable way.

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

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

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

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

2.1.8 AMD SEV

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

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

Thus, AMD provides confidentiality of data by removing trust from the VMM, and creates an isolated environment for the VM to run, where only the VM and the processor can be trusted. However it does not provide integrity of data, allowing replaying attacks to take place, and has a considerably large TCB, since the OS of each VM is trusted. [23]

2.1.9 Discussion

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

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

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

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

2.2 TEE/SGX Enabled Protection Against Untrusted OSes

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

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

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

2.2.1 Virtual Ghost

Virtual Ghost was presented in 2014 [8] as an approach to provide application security against untrusted OSes without requiring higher privileges.

To do so, it introduces the idea of ghost memory which is unaccessible to the OS to read or write, and provides the system with trusted services such as ghost memory management, key management, as well as encryption and signing services.

Virtual Ghost therefore suggests a different design approach. It uses compiler intrumentation to protect the system from external exploits of the OS, by sandboxing and using control-flow integrity and interposes a thin abstraction layer between the kernel and the hardware. Although this layer is similar to a hypervisor, it appears as a library of functions that the kernel code can call directly to interact with the hardware, thus avoiding the need of giving higher privileges than the OS itself.

By using ghost memory and, as said before, preventing the OS from reading and writing ghost memory pages rather than allowing to access this regions of memory, Virtual Ghost makes the system safe from a direct threat from the OS.

Overhead for non-secure data is also prevented since, with that type of data, system calls made by the application don't need encryption or hashing.

Virtual Ghost gives the application flexibility on choosing the encryption and hashing algorithm, so that the system can obtain the desired tradeoff between performance and security.

It is an approach that offers protection from kernel malware, while ensuring comparable performance to TEEs, while adding a layer of protection against external attacks on the kernel itself.

2.2.2 Flicker

Flicker [21] is an infrastructure that provides secured isolated execution of sensitive code while trusting a small TCB, which significantly improves the security and reliability of the code Flicker executes. Although Flicker does not achieve the same level of protection from physical tampering as secure coprocessors do, it provides the same strong guarantees by using modern commodity hardware.

This isolation idea is guaranteed by trusting as few as 250 additional lines of code and, as a result, Flicker circumvents legacy software and eliminates reliance on their correctness, therefore reducing the TCB. Once the TCB has been precisely defined, it's possible for the system to achieve reliability and security.

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

It also provides fine-grained attestation of the code executed to an external party, where the party using Flicker does not leak any information about the state of the system software.

To achieve these properties, Flicker needs hardware that support late launch and attestation, such as AMD and Intel commodity processors, and it's still capable of guarantee them even if the BIOS and OS have become malicious to the system.

2.2.3 MUSHI

MUSHI, or MUltiple level Security cloud with strong Hardware level Isolation, is a framework that provides hardware level isolation and protection to individual guest VMs executing in a cloud infrastructure. It guarantees confidentiality and integrity of a VM even during malicious attacks from both inside and outside the cloud environment.

MUSHI is mainly designed to deal with multi-level security (MLS) [25] systems, therefore MUSHI's main goal is to provide a trusted isolated environment for VM execution. To achieve this, VMs should be instantiated securely and should remain that way throughout their life cycle. With that in mind, MUSHI guarantees the following properties

- Trusted Execution - upon starting a VM, the integrity of both kernel and user image, as well as MUSHI itself, should be attested to the user, thus defining a trusted initial state

- Isolation - VMs running on the same machine should be isolated. As a result, both confidentiality and integrity will be provided to the user VM during execution. However it does not prevent side channel attacks

- User Image Confidentiality - MUSHI provides user image encryption with a key provided by the user itself

MUSHI depends on a very small TCB, including only the hardware, hardware virtualization, BIOS and System Management Mode. It can be implemented using modern commodity hardware containing SMM memory (SMRAM), necessary for the isolation between the host and VM. For remote attestation of VM images, it's done used a TPM. [36]

2.2.4 SeCage

SeCage [20] is an approach that aims to protect user-defined secrets from application potential threats and malicious OS through hardware virtualization, thus isolating sensitive code and critical secrets while denying hypervisor intervention during runtime. It's designed to deal with a strong adversary model, assuming that both application and OS can be compromised, while suporting large-scale software, which usually means a big attack surface.

SeCage devides the system in compartments, where secret compartments have all the functions with permissions to access and manipulate those secrets, and a main compartment is resposible to handle the rest of the code.

The main goals of SeCage are to assure confidentiality of user secrets (e.g., cryptographic keys) and to be a pratical approach with small overheads for large software systems. To achieve this, it ensures

- Hybrid analysis of secrets - provides a mechanism that prevents secrets from being disclosed during runtime, while not affecting their normal usage. For it, SeCage combines static and dynamic analysis. The first one is used to discover potencial functions related to the secrets, while the second one complements the first one, being more precise than the first one. With this, SeCage defines secret compartments, containing a set of secrets and their most common functions. This ensures that only the functions inside a compartment can access that set of secrets.

- Hypervisor protection - isolate each compartment in a completely isolated address space through hardware virtualization, making them unaccessible to the Hypervisor. Only if a function outside a compartment tries to access a secret without permission, it's the Hypervisor job to handle that violation.

- Separating control and data plane - since each compartment needs to communicate with each other, there will eventually be contact with the hypervisor. Therefore, instead of frequent VM exits which cause high overheads, SeCage minimazes hypervisor intervention by only require it to define policies on whether two compartments can communicate (control plane) for as long as they conform to those policies (data plane)

SeCage was shown useful protecting large-scale system software from HeartBleed attacks, kernel memory disclosure and rootkit memory scanning, without adding any significant performance overhead to the application.

2.2.5 InkTag

InkTag [12] is a virtualization-based architecture that protects trusted applications from an untrusted OS, allowing trusted applications to securely use untrusted OS services.

InkTag admits trust in the hypervisor, which is responsible to protect the application code, data, and control flow from the OS, allowing applications to execute in isolation, in a high-assurance process (HAP). Trusted applications can securely and privately share data without interference from the OS or other applications. Each secure application communicates directly with the InkTag hypervisor via hypercalls to detect OS misbehavior.

It introduces a concept called paraverification, which is a technique that simplifies the hypervisor by forcing the untrusted OS to participate in its own verification. As a result, the OS notifies the hypervisor when there is any update to be made to the state, which the hypervisor then checks for correctness. InkTag also ensures virtualization through hardware, which is used to grant isolation, as well as separate secure from insecure data. Access control policies can be specified by each trusted applications to access their secure files, with privacy and integrity of them managed by InkTag through encryption and hashing.

Other important aspect of InkTag is recoverability. InkTag is capable of ensuring crash consistency between file data and metadata. Metadata, which consists in hashed data, is fundamental to guarantee integrity of data therefore, with this consistency granted, InkTag hypervisor can protect the integrity of files even if the system crashes. If for some reason some data is inconsistent, the application must discard it.

2.2.6 Sego

Sego [18] is a hypervisor-based system that gives strong privacy and integrity guarantees to trusted applications, even when the guest OS is compromised or hostile, by removing the trust from the OS. Sego verifies OS services, like the file system, instead of replacing them.

To protect the application from the untrusted OS, Sego relies on a trusted hypervisor and assumes that hardware always executes hypervisor code correctly. It uses paraverification, where the guest OS communicates its actions and intent to the hypervisor, therefore making this verification of the untrusted OS behavior more efficient and easy.

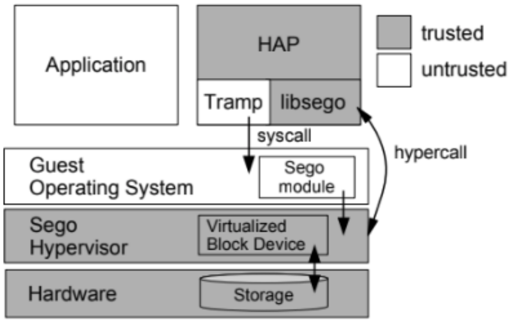

Figure 2.2: Sego Architecture Overview

Sego executes trusted code in a 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 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 OS 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 is used as a way to protect HAP control-flow, since it uses this trampoline as the issuer for the system calls, therefore never compromising the HAP itself. Figure 2.2 shows well enough Sego's overall design without going into to much dept, refering which components are included in the TCB and which are not.

Context switches are handled by the hypervisor, 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.

2.2.7 Other approaches

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

- Hardware-assisted Data-Flow Isolation (HDFI) [34] - data isolation mechanism running on top of RISC-V that uses machine instructions and hardware to enforce isolation, by virtually extending each memory unit with an additional tag that is defined by data-flow. It grants stack protection, standard library enhancement, kernel data protection, virtual function table protection, code pointer protection and information leak prevention. It's easy to use and imposes low performance overhead, while improving security.

- Secure Channel between Rich Execution Environment and Trusted Execution Environment (SeCReT) [16] - it is a framework that is focused in securing the communications between the Rich Execution Environments (REE) and the TEE built in ARM TrustZone, to add to the idea of isolation from the OS. It enables legitimate processes to use a session key in the REE, which is regarded as unsafe. To protect the key, SeCReT verifies the code's integrity and control-flow of the process every time a switch between user mode and kernel mode takes place. SeCReT's key-protection mechanism is only activated during the runtime of the process that has permission to access TrustZone, so it minimizes the performance overhead.

2.2.8 Discussion

Another conclusion about the differences of the approaches.

//TODO

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

Protection against untrusted OSes.

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

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

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

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

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

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

2.3 SGX-Frameworks and Application Support

The need for cloud computing is constantly growing in modern applications. It is a cost-effective and pratical solution to run large distributed applications, however the fact that it requires users to trust the cloud provider with their code and data creates some trust concerns for developers. Although the usage of TEEs aim to tackle this problem by running and storing sensitive data on a isolated environment, protecting that data from unauthorized access. To accomplish this, the main approach is to devide the application into trusted and untrusted parts, reducing the TCB as much as possible as a way to reduce security breaches. In the next sections we'll discuss frameworks that can accomplish solutions to the problem described above, by working with Intel-SGX as the TEE provider, as a way to complement its regular execution.

2.3.1 VC3 protection for MapReduce Jobs

Verifiable Confidential Cloud Computing (VC3) [29] is a framework that achieves confidentiality and integrity of data, as well as verifiability of execution of code with good performance through MapReduce [10] techniques. It uses Intel SGX processors as a building block and runs on unmodified Hadoop [33]. In VC3 users implement MapReduce jobs, compile and encrypt them, thus obtaining a private enclave code E-. They then join it with a small portion of public code E+, that implements the protocols for key exchange and job execution. Users then upload the resulting binary code to the cloud, where enclaves containing both E- and E+ are initialized by an untrusted framework F.

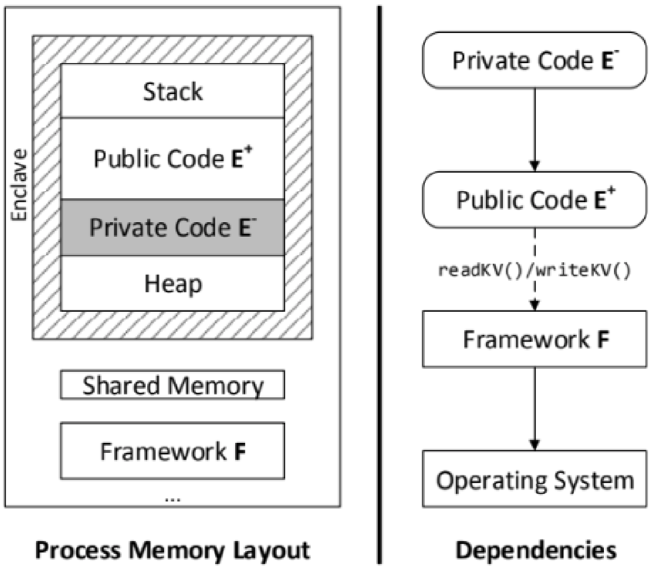

Figure 2.3: VC3 memory design model on the left, component dependencies on the right

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

2.3.2 Protected ZooKeeper

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

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

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

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

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

2.3.3 Ryoan Sandboxing

Ryoan [14] consists on a distributed sandbox that allows users to protect the execution of their data. This is achieved with the help of Intel SGX [15] [22] enclaves, creating sandbox instances that protect data from untrusted software while also preventing leaks of data, which is a weakness of enclaves caused by side channel attacks. Ryoan does not include any priviledged software (e.g. OS and hypervisor) in it's TCB, while trusting only the hardware (SGX enclave) to assure secrecy and integrity of the data.

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

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

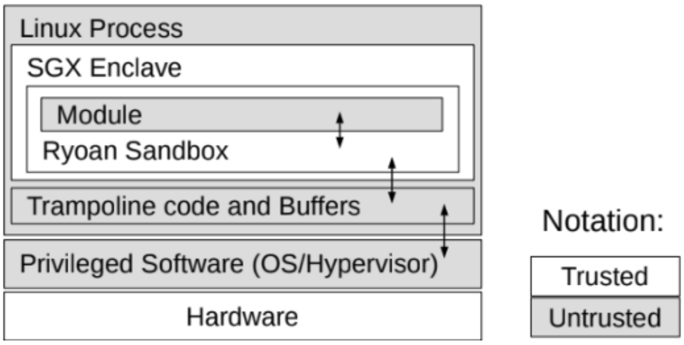

Figure 2.4: Instance of Ryoan running on a single machine

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

2.3.4 Opaque

Opaque [37] is a distributed data analytics platform that guarantee encryption, secure computation and integrity to a wide range of queries. Therefore, instead of being implemented in the application layer or the execution layer as this kind of security approaches usually are, Opaque is implemented in the query optimization layer.

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

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

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

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

2.3.5 Graphene-SGX

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

Graphene-SGX [35] came to help circunvent those restrictions, while still assure security to the data. It is a library OS that aims to reproduce system calls, respecting security concerns, so that unmodified applications can use them to keep executing normally without interacting directly with the OS or hypervisor.

By using a library OS, the system is expected to lose performance and, since a new layer of software was added, to increase the size of the TCB. Although these assumptions are true, they are quite often exagerated. Graphene-SGX's performance goes from matching a Linux process to less than 2x in most executions of single-processes. Graphene-SGX has also shown some great results comparing it to other similar approaches that use shim layers, such as SCONE [1] and Panoply [32], where it shows to be performancewise similar to SCONE and faster 5-10 percent than Panoply, while adding 54k lines of code to the TCB compairing to SCONE's 97k and Panoply 20k.

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

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

2.3.6 Network services protection approaches

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

In [17] it's shown that leveraging hardware protection of TEEs can grant benefits, such as simplify the overall design of the application, as well as securely introduce in-network functionality into TLS sessions. The same paper also presents a possible approach to reach security and privacy on a network level, by building a prototype on top of OpenSGX, that shows that SGX-enabled applications have modest performance losses compared to one with no SGX support, while significantly improving it's security and privacy.

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

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

In [31] by implementing the S-NFV approach with Snort [27] on top of OpenSGX was concluded that this SGX-enabled approach results in bigger overheads (aprox. 11x for gets and 9x for sets) than an SGX-disabled Snort application, at the cost of extra security.

2.3.7 Application-level protection approaches

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

In addition to Graphene-SGX 2.3.5, approaches like Haven [3], Scone [1] and Panoply [32] offer system support, by ensure a secured way for the application to make system calls, such as implementing a library OS or a standard library, inside the enclave. Haven runs an entire library OS (LibOS) inside the enclave, resulting on a very large TCB. Scone uses sandboxing as a way to reduce TCB size due to the LibOS approach. Panoply provides the abstraction of micro-container (micron), having only to import specific micron-libraries instead of the whole LibOS, resulting in a shorter increase of the TCB size.

Although capable of ensuring unmodified applications a way of running on top of SGX, the increase of the TCB size has been seen as a possible vulnerability. Thus new approaches more focused on this TCB size problem, like Glamdring [19] and SGX-Shield [30], have been developed lately by the community as possible alternatives to the above ones.

2.3.8 Discussion

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

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

SGX frameworks and applications.

VC3 [45] runs MapReduce jobs in SGX enclaves.

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

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

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

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

2.4 Related work analysis and rational

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

CHAPTER 3

ELABORATION WORK DIRECTIONS

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

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

3.1 Materialization of objectives and contributions

3.2 System Model and Reference Architecture

3.3 Prototyping effort

3.4 Prototype Validation and Experimental Assessment

3.5 Open issues

CHAPTER

4

ELABORATION PLAN

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

BIBLIOGRAPHY

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

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

[3] A. Baumann, M. Peinado, and G. Hunt. "Shielding applications from an untrusted cloud with Haven." In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*. Best Paper Award. USENIX - Advanced Computing Systems Association, 2014. URL: `https://www.microsoft.com/en-us/research/publication/shielding-applications-from-an-untrusted-cloud-with-haven/`.

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

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

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

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

[8] J. Criswell, N. Dautenhahn, and V. Adve. "Virtual Ghost: Protecting Applications from Hostile Operating Systems." In: *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS*

14. ACM Press, 2014. DOI: 10.1145/2541940.2541986. URL: https://doi.org/10.1145/2541940.2541986.

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

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

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

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

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

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

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

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

[17] 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.

[18] 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.

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

[20] 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`.

[21] 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`.

[22] 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`.

[23] 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`.

[24] 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`.

[25] *Red Hat Documentation - Multi-Level Security (MLS)*. Accessed: 26-06-2019. URL: `https://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/sec-mls-ov.html`.

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

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

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

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

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

Appendix A

Appendix 1 Lorem Ipsum

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

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

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

pellentesque felis eu massa.

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

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

Appendix B

Appendix 2 Lorem Ipsum

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

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

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

pellentesque felis eu massa.

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

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

Annex I

Annex 1 Lorem Ipsum

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

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

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

pellentesque felis eu massa.

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

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