Miguel de Lemos Dias Rosa Anciães

Bachelor of Computer Science and Engineering

A Trusted and Privacy-Enhanced In-Memory Data Store

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

Master of Science in
Computer Science and Engineering

Adviser: Henrique João Lopes Domingos,
DI-FCT-UNL, NOVA LINCS

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2020

Abstract

The recent advent of hardware-based trusted execution environments provides isolated execution, protected from untrusted operating systems, allowing for the establishment of hardware-shielded trust computing base components. As the processor provides such "shielded" trusted execution environment (TEE), their use will allow users to run applications securely, for example on the remote cloud servers, whose operating systems and hardware are exposed to potentially malicious remote attackers and non-controlled system administrators' staff. On the other hand, Linux containers managed by Docker or Kubernetes are interesting solutions to provide lower resource footprints, faster and flexible startup times, and higher I/O performance, compared with virtual machines (VM) enabled by hypervisors. However, these solutions suffer from software kernel mechanisms, easier to be compromised in confidentiality and integrity assumptions of supported application data. This dissertation will design, implement and evaluate a Trusted and Privacy-Enhanced In Memory Data Store, making use of a hardware-shielded containerised OS-library to support its trust-ability assumptions. To support large datasets, requiring data to be mapped outside those hardware-enabled containers, our targeted solution will use partial homomorphic encryption, allowing trusted operations executed in the protected execution environment to manage in-memory always-encrypted data, that can be or not mapped inside the TEE.

Keywords: Hardware Security; Privacy-Enhanced Data Store; Homomorphic Encryption; Isolated Environments; Trusted Computing; Cloud Computing; Virtualisation; Containerisation; Availability; Reliability.

Resumo

Os recentes avanços de ambientes de execução confiáveis baseados em hardware fornecem execução isolada, protegida contra sistemas operativos não confiáveis, permitindo o estabelecimento de componentes base de computação de confiança protegidos por hardware. Como o processador fornece esses ambientes de execução confiável e "protegida"(TEE), o seu uso permitirá que os utilizadores executem aplicações com segurança, por exemplo em servidores *cloud* remotos, cujos sistemas operativos e hardware estão expostos a atacantes potencialmente maliciosos assim como administradores de sistema não controlados. Por outro lado, os *containers* Linux geridos por sistemas *Docker* ou *Kubernetes* são soluções interessantes para poupar recursos físicos, obter tempos de inicialização mais rápidos e flexíveis e maior desempenho de I/O (interfaces de entrada e saída), em comparação com as tradicionais máquinas virtuais (*VM*) activadas pelos hipervisores. No entanto, essas soluções sofrem com software e mecanismos de kernel mais fáceis de comprometerem os dados das aplicações na sua integridade e privacidade.

Esta dissertação projectará, implementará e avaliará um Sistema de Armazenamento de Dados em Memória Confiável e Focado na Privacidade, utilizando uma biblioteca conteinerizada e protegida por hardware para suportar as suas suposições de capacidade de confiança. Para oferecer suporte para grandes conjuntos de dados, exigindo assim que os dados sejam mapeados fora dos *containers* seguros pelo hardware, a solução planeada utilizará encriptação homomórfica parcial, permitindo que operações executadas no ambiente de execução protegido façam gestão de dados na memória que estão permanentemente cifrados, estando eles mapeados dentro ou fora dos *containers* seguros.

Palavras-chave: Segurança de Hardware; Armazenamento de Estrutura de Dados em Memória Confiável e Focado na Privacidade; Encriptação Homomórfica, Ambientes Isolados; Computação Segura; Computação em *Cloud*; Virtualização, Containerização; Disponibilidade; Confiabilidade.

Contents

List of Figures

Listings

Acronyms

ACL	Access Control List
AIK	Attestation Identity Key
API	Application Programming Interface
AWS	Amazon Web Services
CA	Certification Authority
DBMS	Database Management System
DDoS	Distributed Denial of Service
DoS	Denial of Service
Ecall	Enclave call
EK	Endorsement Key
EPC	Enclave Page Cache
EPCM	Enclave Page Cache Mapping
GB	Gigabyte
HSM	Hardware Security Module
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
IaaS	Infrastructure as a Service
IoT	Internet of Things
KVS	Key-Value Store
LRU	Least Recently Used
LSM	Log-Structured Merge Tree

MB	Megabyte
MIM	Man-in-the-middle
Ocall	Out call
OS	Operating System
OTP	One-Time Password
P2P	Peer to Peer
PCR	Platform Configuration Register
PRM	Processor's Reserved Memory
RAM	Random Access Memory
REST	Representational State Transfer
RSA	Rivest–Shamir–Adleman
SaaS	Software as a Service
SASL	Simple Authentication and Security Layer *
SDK	Software Development Kit
SGX	Software Guard Extensions
SQL	Structured Query Language
SSL	Secure Sockets Layer
syscall	System call
TB	Terabyte
TCB	Trusted Computing Base
TCE	Trusted Computing Environments
TCG	Trusted Computing Group
TEE	Trusted execution environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
USB	Universal Serial Bus
VM	Virtual Machine

Chapter 1

Introduction

In this chapter it's presented the context and motivation for this thesis, the main problem statement followed by the goals and objectives and all the planned contributions. In the end, it is presented the structure used in the following chapters of the document.

1.1 Context and Motivation

Cloud computing has gone through many steps that include grid and utility computing, application service provision and software as a service before reaching the level we know these days. The concept of delivering continuous resources through a global network is rooted in the 1960's. Some experts credit the professor and computer scientist John McCarthy [62] be proposing the concept of computation being delivered as a public utility.

Then, around 1970's the concept of the virtual machine (VM) started to gain popularity as it permitted multiple distinct computing environments to reside on one physical machine.

One of the first major cloud computing moments was the arrival of *salesforce.com* that pioneered the concept of delivering enterprise applications via a simple website. Later, around the 2000's, current big names like Oracle, SAP, Google, Amazon and Microsoft joined the trend and made the cloud world as it is today [2, 3].

Over the past decade, cloud computing has evolve from something service providers told companies they should adopt, to becoming the technology heart of not only major companies, but medium sized enterprises, small start-ups, personal projects and pretty much anyone who works in the computer science world.

Recent studies are foreseeing that 83% of enterprise workloads will be on the cloud by 2020 [1]. The array of services provided now are endless and the costs are attractive

to businesses. These services allow developers to only pay for resource usage, and to take advantage of all the power of very large companies. Scalability at request, reliability with daily backups and seamless integration with a lot of other services are some advantages of moving to the cloud. And all of these functionalities without having to manage big infrastructures and a lot of servers, networks, disks, etc... [77].

All of this data and processing happening in someone else's machine started to raise privacy and security concerns. It has become a very attractive target for malicious hackers to attack cloud providers due to the amount of data they process and hold on their services. The best security researchers are always working with the providers to try and mitigate all bugs and vulnerabilities on their very large platforms which has become also a big attack vector. It has been reported by Microsoft, that *"There was a 300 percent increase in Microsoft cloud-based user accounts attacked year-over-year (Q1-2016 to Q1-2017)."* and also *"The number of account sign-ins attempted from malicious IP addresses has increased by 44 percent year over year in Q1-2017."* [57]. Another example published on the Washington Post describes a sophisticated Man-in-the-Middle (MIM) cyber-attack that has targeted Apple's iCloud service in China, in an apparent attempt to collect user names, passwords and other private information [9]. Also, Amazon Web Services has been in 2019 hit by a massive DDoS (Distributed Denial of Service) attack that kept the system down for about 8 hours straight, which can mean thousands of dollars lost by clients [17].

The use of virtual machines to lodge different computing environments on the same physical machine can also raise problems, as explained by the publication *"Seriously, get off my cloud! (...)"* where the researches were able to exploit and obtain RSA (encryption) keys from other VMs deployed in the same physical machine of Amazon EC2 service. This work affirms the need for stronger isolation techniques in public clouds [40].

Docker containers and Kubernetes clusters, used instead and/or alongside traditional VMs, are two of the most popular technologies among cloud providers and cloud server environments these days, and if not managed correctly can become attack vectors. They are already being exploited, most known by the report of Tesla Motors [76], suffering a breach because of an exposed Kubernetes instance [24, 25].

The well known Cambridge Analytica scandal [78] gave the world another perspective about the the security guaranteed by the cloud providers, social media and every platform that keeps user's data in a non secure manner. It shows how it can be exploited, sold and manipulated without the owner's consent.

As for Redis as a storage solution, and explained in depth in section 2.1.2, *"Redis should not be publicly exposed as it has no default authentication and all the data is stored in clear text"* and so, according to some studies, there are around seventy two thousand Redis servers available online today, and over 75% of them were compromised and infected with some kind of malware [14, 26, 59].

Its certain that as the cloud environment grows, the motivation for malicious hackers to attack and try to steal information will grow with it, and with all these security

breaches increasing year after year, a previously mentioned study also reflects that *"66% of IT professionals say security is their greatest concern in adopting an enterprise cloud computing strategy"* [1].

1.2 Problem Statement

The problem behind the goals and objectives of this dissertation can be summarised in designing a system, answering the following questions:

Is it feasible to implement a solution for a remote trusted and privacy-enhanced cloud-based key value store system providing strong security and privacy features in a trustworthy solution? Can we design and implement those features with a good trade-off with operational and performance criteria under scalability and reliability guarantees? Can we address the trust-ability by minimising the trust-computing base using protected components running as isolated trusted bases in hardware-shielded trust execution environments? Is it possible to remove the threat of administrators of a cloud platform breaking data privacy? Can we combine the security guarantees with privacy-enhanced in-memory operations supporting big data sets and grained data-structures?

1.3 Objectives and Planned Contributions

The main goal of this dissertation is to design implement and evaluate a privacy enhanced in-memory key-value store to be used as a trusted cloud service with hardware based security features running in a trusted execution environment supported by Intel's SGX technology. To overcome the protected memory limitations and coarse-grained paging as basically supported in SGX memory-management facilities, our solution must support flexible big data sets in a hybrid approach using SGX-mapped protected memory and unprotected memory. Data in unprotected memory will be encrypted and operated in the encrypted form, using partial homomorphic encryption techniques. For implementation purposes we will address our solution to be leveraged from the REDIS technology, enhanced in a architecture using isolated and containerised services running in isolated Intel-SGX trusted-execution-enclaves but supporting all the variations of REDIS-based architectural deployments, e.g.: as a single REDIS server instance, or using replicated instances (with master-slave and master-slave tree-chains, as well as, clustered instances). We will analyse and compare the designed solution in terms of the introduced security benefits and measuring the overheads introduced by the additional security, privacy and trust-ability guarantees.

In this thesis we plan to achieve the following contributions:

- **Design and implementation of the cloud-enabled privacy-enhanced solution**, with all-in-the-box planned features as described above, and able to be used as a "cloud-platform as a service" solution, providing:

  - **Trust-ability, security and privacy** properties with the following guarantees: high availability, built-in replication, LRU eviction model, in-memory operations on encrypted big data sets and complementary options for protected on-disk persistence.
  - **Software attestation** guarantees provided to clients (users) in order to validate the correctness and integrity of the remote software stack providing the solution.

- **Multiple Replication Mechanisms** based on the same secure solution to analise how these types of replication (centralised solution, a Master-Slave architecture and a clustering solution) will be impacted by the additional security features.

- **Drastically reduce TCB** in the remote cloud provider by removing the millions and millions of lines of code implementing the hypervisors and operating systems used in their infrastructures thus creating a **truly isolated system** by leveraging Intel's SGX technology to create a shielded and trusted execution environment in a remote cloud provider.

- **Complete analysis report** of the different solutions of replication and security levels, comparing a normal non-secure solution with the the privacy-enhanced implementation along with evaluation of overheads and trade-offs introduced by the additional security mechanisms.

1.4 Report Organisation

The remaining of the report is organised as follows:

Chapter two presents the topic background, related work and initial research performed for this thesis, including relevant contributions and similar solutions existing in current days.

Chapter three will discuss the approach to the elaboration phase by describing the planned system architecture and technologies that will be used. It provides a in-depth explanation of how its planned to achieved the goals and objectives of this dissertation.

Chapter four provides a planned timeline to be followed throughout the elaboration of this thesis including a breakdown of the work-plan by weeks and categories from the beginning to the thesis delivery and presentation.

Chapter 2

Related Work

This chapter presents and briefly discusses the related work and the study performed beforehand in order to guide and give some context to the reader. It will present work that was used as the basis of this thesis, existent technologies and their relation with this project, and some comparisons between those existing technologies, the problem addressed in this thesis and the solutions proposed to solve, or better address, those very same problems.

First, in section 2.1 we explain and discuss for the first time the definition of a Key-Value Store. We present some use cases, current technology available, their differences and most importantly their security models and concerns. Having discussed the software, section 2.2 will then address the environment on where that software will run on, most specifically the hardware. It explains and present the different ways to secure and authenticate the hardware, prevent hardware-based attacks and discuss some of the current products available and how they will be used across this thesis. Section 2.3 will then make the bridge between software and hardware. It explains how Key-Value stores are currently being run on secure environments. It discusses how software and hardware work together to achieve a secure application. This chapter will be focused on the Intel SGX secure model and explain the advantages and disadvantages of this module. To conclude the chapter, section 2.4 will combine the information of every sub-chapter and analyze it with a bigger perspective and better knowledge of the theme.

Section 2.3 is considered to be the **main core** investigation and directly related to the work planned for this thesis. As for the other sections, they provide a background knowledge necessary for understanding of the core of this dissertation.

Along the next chapter we summarise the main relevant ideas that can be retained from each section for our objectives and expected goals.

2.1 Key-Value Stores

Key value stores are the simplest form of what computer scientists call a database. The simplicity lies on associating a value to a certain key and storing that pair, as well as retrieving the values of known keys. [44]

Listing 2.1: Redis Set & Get

```
redis> SET mykey "Hello"
"OK"
redis> GET mykey
"Hello"
```

Is this simplicity that makes this technology very attractive to developers. The ease of use, its high performance and speed are key aspects in favour of this technologies. However, simply working with keys and values might not be enough to more complex applications, and that is why Key-Value store product developers are introducing new features in order to make them appealing to a broader mass of users, always keeping them lightweight and fast.

For that lightweight and fast attributes, most of the key-value stores work in the computer memory. This allows fast get and write operations as opposed to persistent disk storage. Although they work mainly in memory, most of the solutions offer some persistent mechanism so we can make use of its performance but still persist data in case of a disaster, server failure or any crash.

KVSs have been evolving for years and some are now more than a single key-value store module. A lot of them are now supporting a multi-model storage. Meaning that a value can be more than a single integer or a string. For example, Redis [63] as a multi-model store is not only a key-value store, but also [64]:

- **Document Store** - *"nonrelational database that is designed to store and query data as JSON-like documents"* [29]
- **Graph DBMS** - *"Graph databases are purpose-built to store and navigate relationships. Use nodes to store data entities, and edges to store relationships between entities"* [33]
- **Search Engine** - *"nonrelational database that is dedicated to the search of data content. Use indexes to categorize the similar characteristics among data"* [70]
- **Time Series DBMS** - *"Provides optimum support for working with time-dependent data. Each entry has a timestamp, the data arrives in time order and time represents a primary axis for the information."* [79]

So, the KVS world is becoming more and more versatile as the years pass.

In the next subsections its discussed and presented the overview of the current KVS technology. We picked the some top KVSs technologies nowadays according to db-engines [45] website.

2.1.1 Memcached

Memcached [52] is a free and open source key-value store released in 2003. It is described as a high performance distributed memory object caching system.

It is design to hold small chunks of data (strings and objects) to work as a cache for results of database calls, API calls, or page rendering. Its biggest use case is for use in speeding up dynamic web applications by alleviating database load.

This system lies on the simpler key-value store spectrum. It takes advantages of the simplicity of a key-value store to edge ease of development, and solving many problems facing large data caches. Its API is available for most popular languages. It has a LRU eviction technique which means that items will expire after a specified amount of time if not used.

When it comes to system replication, availability and reliability, Memcached has an interesting approach. In order to keep it blazing fast, there is no communication between server instances in a cluster. Memcached servers are unaware of each other. There is no crosstalk, no synchronization, no broadcasting, no replication. Adding servers will only increase the available memory.

As for its security context, Memcached spends very little, if any, effort in securing the systems for random internet connections. The servers only have support for SASL [67] authentication mechanism. This method of authentication is not implemented as end-to-end encryption, it only provides restriction access to the daemon, but it does not hide communications over the network. That means it is not meant to be exposed to the internet or to any untrusted users [53].

2.1.2 Redis

Redis [63] is an in-memory data structure store that can be used as a database, cache and also a message broker. Redis focuses on performance, so most of its decisions prioritize high performance and very low latency.

It has been benchmarked as the world's fastest database [65] and together with a their multi-model and its rich set of operations that can be performed over data it has been the leading key-value store according to use and popularity for a multiple set of years [45].

Listing 2.2: How Fast is Redis

```
$ redis-benchmark -t set -r 100000 -n 1000000
====== SET ======
1000000 requests completed in 8.78 seconds
50 parallel clients
3 bytes payload
keep alive: 1

99.59% <= 1 milliseconds
99.98% <= 2 milliseconds
```

```
100.00% <= 2 milliseconds
113934.14 requests per second
```

As said before, Redis is now not a simple KVS. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. It also has built-in replication, server side scripting, LRU eviction, concept of transactions and different levels of persistence. It provides high availability and automatic partitioning as well.

Redis provides replication in form of a master-slave model. This form of replication works with a single node (master node) where all writes occurring will be replicated to the other Redis instances (slave nodes). Writes on nodes other than the master will not be replicated. Redis provides a read-only setting that can be applied to slave nodes to prevent states differences between instances.

Security is not Redis' primarily concern (just like others). *"In general, Redis is not optimised for maximum security but for maximum performance and simplicity"* [66]. It is design to be access by trusted clients inside trusted networks. This means that it is not supposed to be publicly exposed. Redis implements a simple authentication system with a password on the configuration file for client authentication. It is also advised to run it behind a proxy to enable some ACL policies and SSL network security.

There are a few other security concerns that Redis addresses, but has we can now start to see, in this types of stores, security falls behind performance and usability.

2.1.3 Amazon Dynamo DB

Amazon Dynamo DB [6] is a fully managed NoSQL database service. It is a key-value store and a document store that is built based on the dynamo paper [27]. This paper describes a P2P (peer-to-peer) network with high availability, eventual consistency and very easily scalable. It also successful handles server and data center failures and network partitions.

Amazon builds on this paper and offers DynamoDB as a service in their platform. It is a hosted system in the Amazon Web Services [8] infrastructure and it is fully managed. That means no need for low level server configurations or maintenance. It is all managed by the AWS team and offered to the user with a nice configuration interface. It also means that it has built-in security, backup and restore and in-memory caching for internet-scale applications. It also offers seamless scalability by increasing the number of nodes/servers according to current traffic received by the application on any given time.

This technology focuses more on high availability but also achieves very high performances and very low latency and being fully managed it also takes advantages of the AWS infrastructure full power. It currently sits second on the db-engines [45] most popular ranking.

2.1.4 Microsoft Azure Cosmos DB

Microsoft Azure Cosmos DB [55] is a fully managed database service provided by Microsoft Azure [56]. This service provides a global distributed, horizontally scalable, multi-model database. Its multi-model architecture can work as a key-value store, a Document Store, a graph DBMS and a wide column store.

It's very proud and excels in the easy of global scale with the system call *Turnkey global distribution*, providing transparent multi-master replication and a set of users configurable consistency options. It also strongly advertises a *Multi-Model Multi-Api* feature where you can use multiple data types on this single database service. Cosmos DB automatically indexes all data and allows the user to use various NoSQL APIs to query the data.

As a fully managed service, Cosmos DB makes use, in the background, of the large infrastructure with almost unlimited resources and capabilities provided by Microsoft, which means it also has built-in security, fail-over mechanisms for disaster recovery, and high performance with single digit read and write latencies.

2.1.5 Microsoft Azure Cache for Redis

Microsoft Azure Cache for Redis [54] is a service provided by Microsoft Azure that joins the open source world of Redis with the commercial side of a fully managed and hosted platform.

It uses at its core the Redis server technology and provides ease of deployment and management, built-in global replication, Azures' infrastructure security and flexible scaling and Redis superior throughput and low latency performance.

Being in the Azure ecosystem provides nice integration with all Azures' services as shown in figure 2.1.

2.1.6 Aerospike

Aerospike [4] is an enterprise-grade, high performance Key-Value Store. It is another KVS technology currently available today. It promises a philosophy of *"no data loss"* through Strong Consistency. Normal systems trade requiring this type of consistency usually trade performance for data integrity but Aerospike allows it with minimal performance loss. That means it can be used for example in banking payments, retail and telecommunications use cases.

It also provides a dynamic cluster management and unique flexible storage. That enables very easy deployments and particularly very easy scalability, so it is able to meet any data volume needs and still maintaining low latencies across that wide range of data volumes, from low volumes until hundreds TB of data.

As for security, it includes (the enterprise version) a database access management and audit trail logs. It also includes transport level encryption for client-server traffic

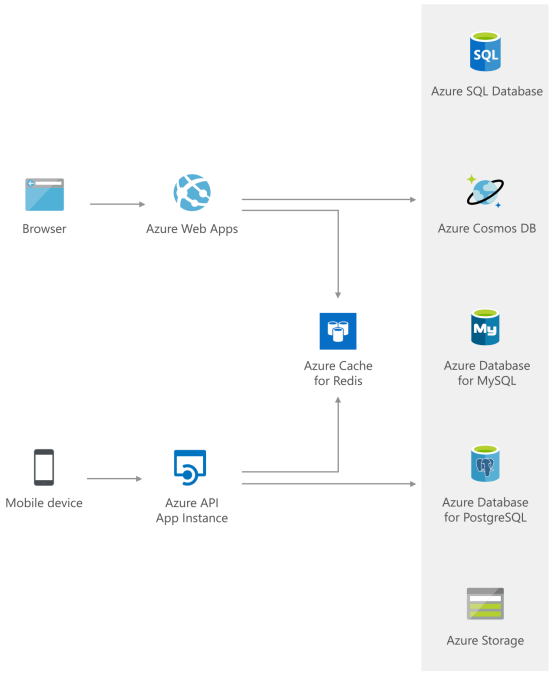


Figure 2.1: Azure Environment Integration

and cross-datacenter traffic [5].

2.1.7 Discussion

In this chapter when gather information about the overview of th current Key-Value Store. We can conclude that the most important feature of this technology is the performance and all of the above products mentioned do focus on that characteristic. Some of them even compromise in another features to achieve the best performance possible. Security is not the main concern and the most used measures in the current technologies being securities implementations at the network and transport level by using TLS and also full disk encryption.

Network and transport layer security is a must when implementing any system, and this thesis will also use those standards.

As for full disk encryption on the server, it opens up some attack vectors. Full disk encryption means that random users will not be able to query the data but credentialed users can. Although, anyone with full access to the database, for example database operators or/and administrators, can decrypt and access all information. This creates a risk of privacy breaking due to hackers wielding stolen credentials, rogue insiders who have been granted more access than they need or the well known honest-but-curious adversary model, where an administrator with full credentials does not have bad intentions, but, driven by curiosity, access information therefore breaking data privacy. A cloud based KVS service like the ones talked above, this type of vulnerabilities can be a major concern for a use case with very sensitive data since the server would be off premises, there

is no control over it when it comes to privacy of data.

This thesis will implemented a system based on Redis, the most popular and used Key-Value Store currently used and will try to solve some of the problems with security described above. It will compare the principle feature of a KVS, the performance, of a simple and normal Redis server and a privacy-enhanced Redis solution so the user can calculate the trade-off between performance and security and applied the correspondent solution to their own use case.

2.2 Trusted Computing Environments

Modern data processing services hosted in the cloud are under constant attack from malicious system administrators, server administrators and hackers who exploit bugs on applications, operating systems or even the hypervisor. However, current days shows a massive trend of business moving to the cloud infrastructure looking for easy deployment, managed services with built-in replication and fault tolerance, fast and trivial scaling and predicted costs.

With more and more data exposed in the cloud, hackers have a bigger desire to exploit and look for vulnerabilities. This results in frequent data breaches that reduce trust in online services. The need for cloud providers to ensure a level of security and trust to make the user comfortable of moving its data to the cloud has never been bigger, and with that need some solutions in the form of Trusted Computing Environments (TCE) appeared.

Trusted Computing is a concept that strives to provide strong confidentiality and integrity guarantees for applications running on untrusted platforms. It forces a certain machine to behave an expected way even if running on a remote or machine that is out of our control.

TCE will also provide a decrease of the Trusted Computing Base (TCB) - the amount of components that the application needs to trust in order to run smoothly. By isolating the service running on this trusted environments (limiting the set of instructions available and encrypting data), it prevents the operating system, the hypervisor and even malicious system administrators (three components normally on the TCB) to break data confidentiality and integrity within this environments.

There are a few hardware/software based solutions to achieved a trusted computing environment, and they will be explained in the next sections.

2.2.1 TPM – Trusted Platform Modules

A Trusted Platform Module, also known as a TPM is a technology proposed by the Trusted Computing Group (TCG) designed to provide hardware-based security related functions. It's a chip embedded into the motherboard and includes multiple security

mechanisms to make it tamper resistant to physical harm and malicious software is unable to mess with its security features [80]. Some key advantages of using TPMs are:

- Generate, store, and limit the use of cryptographic keys
- Platform identity by using the TPM's unique RSA key, which is burned into itself also known as Endorsement Key (EK) and never leaves the TPM.
- Help ensure platform integrity by taking and storing security measurements.

Figure 2.2 shows the mains components and services provided by a TPM module. As shown in the figure, all of them only have one access point I/O which is protected and safely managed by the TPM execution engine.

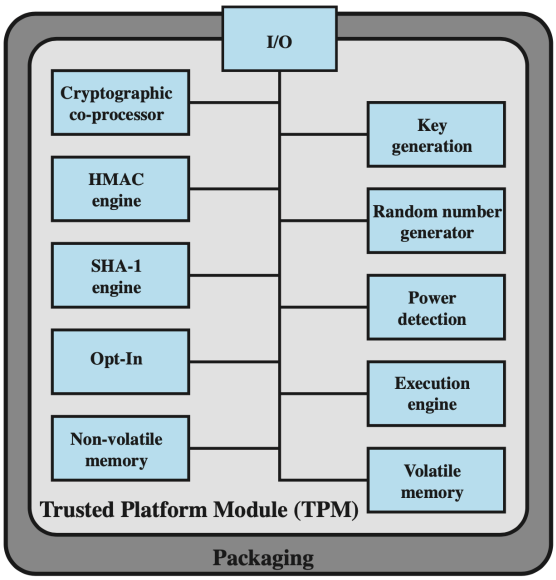


Figure 2.2: TPM insides

With all the components described by figure 2.2, TPMs provide there main TPM features: Encryption, Authenticated Boot and Attestation.

The first feature is used for every security and confidentiality aspects, mainly generating cryptographic keys, encrypting, signing and hashing data with secure standard algorithms melted in the module.

Authenticated Boot is the ability to boot the OS in stages, assuring that each portion of OS, as it is loaded, is a version trusted and approved for use, detecting hardware and software changes on every stage to verify if the code loaded can be trusted. This boot sequence happens with the help Platform Configuration Registers (PCR) that store the trusted software hashes.

The attestation feature is a way for a client to remotely check the state of a machine and will be further explained in the next subsection.

2.2.2 TPM - Enabled Software Attestation

The remote attestation feature of a TPM is the ability of a program to authenticate itself against external verifiers. Is a mechanism that allows a remote party to verify the internal state of the OS or another software and decided whether or not that piece of software is intact and trustworthy. The verifier can trust that the attestation data is accurate and not tampered with because it is signed by the internal key of the TPM, a special key known as the Attestation Identity Key, known from now on as AIK [16].

A remote attestation procedure is described in image 2.3 [15]:

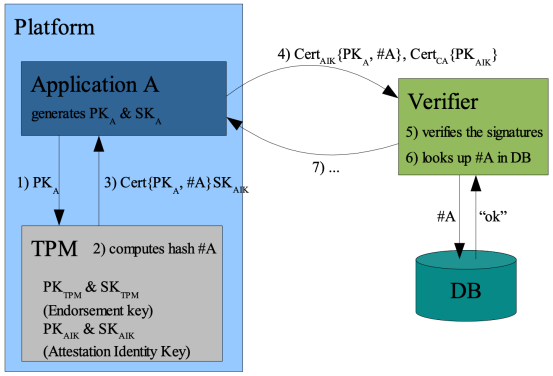


Figure 2.3: Remote Attestation Procedure

1. The application "A" generates a public/private key pair PK_A & SK_A and asks the TPM to certify it.
2. The TPM computes a hash value #A of the executable code of program "A".
3. The TPM creates a certification including PK_A and #A and signs it with the attestation identity key SK_{AIK}.
4. When application "A" wishes to authenticate itself to a remote party, it sends the cert. of its public key and hash value #A along with a cert. issued to the TPM by a trusted certification authority (CA).
5. The remote party verifies the cert. chain.
6. The remote party looks #A up in a database which maps hash values to trust levels.
7. If application "A" is deemed trustworthy, we continue the communication, probably by using PK_A to establish a session key.

2.2.3 HSM – Hardware Security Modules

An (HSM) hardware security module is normally an external module that can be added to a system, in form of a USB device or a component living in a secure network as a

trusted server, instead of being embedded into the motherboard like a TPM. It provides a dedicated system of hardware enable accelerated cryptographic functions like encryption, decryption, key generation and signing capabilities [38]

What makes this devices so secure, like the TPM, is it can´t be interfered with by external code, and it provides an array of protective mechanisms to detect and prevent external physical tampering like drill protection foil, resin-embedded chips as well as temperature and voltage sensors. Any detection of tampering will result in an alarm as well as countermeasures by the applications installed inside. [21].

HSM can have various applications and can be used in simple forms for example a specific bank dongle that generates OTP (one-time password) for accessing your account or be a big corporation and enterprise appliance in various industries, e-health, automotive and IoT systems.

2.2.4 Trusted Execution Environments

A Trusted Execution Environment (TEE) is an abstraction that describes a machine capable of executing a given program P in isolation, i.e. whose output is determined by the initial state of P and a set of defined inputs given into the TEE (Barbosa et al., 2016).

It is a secure area of the main processor that ensures sensitive data and code loaded inside is stored, processed and protected in an isolated and trusted environment. As such, it offers protection from software attacks even the ones generated in the operating system.

A TEE guarantees that:

- The code loaded in the environment is authentic and was not tampered by an attacker.
- All system state is correct (CPU registers, memory and sensitive I/O).
- The code, all data generated and runtime state is confidential and stored persistently.

The threat model of a TEE should include all software attacks and the physical attacks performed on the main memory and its non-volatile memory.

"There are many interpretations of what is meant by Trust. In the TEE it is used to imply that you may have a higher level of trust in validity, isolation and access control in items (assets) stored in this space, when compared to more general purpose software environments"[84].

2.2.5 Intel SGX

"Intel® Software Guard Extensions (Intel® SGX) is a set of instructions that increases the security of application code and data, giving them more protection from disclosure or modification."[41].

These set of instructions are one of the latest iterations of trusted computing solutions and designs that tries to tackle the problem of securing remote computations by leveraging secure hardware on the remote host machine. The SGX processor enables a secure container called enclave which protects the confidentiality and integrity of the execution, such as code and data while relying on software attestation mechanisms.

A SGX can be though as a reverse sandbox. With a sandbox you are trying to protect the system from your application, but with SGX you are trying to do the opposite and protect the application from the system. The system can be the OS, the hypervisor, the BIOS, the firmware or even the drivers [72].

A SGX enabled application is broken into two parts, the untrusted and trusted parts. The trusted part of the application is all the processing that deals with any sensitive data the application is handling. This part will be run inside enclaves and be stored in protected memory. The rest will live in normal memory and not be protected.

It provides this kind of security from the hardware by isolating all the private data from the outside, placing it into a restricted area of the memory called the PRM more specifically in the EPC (Enclave Page Cache) as shown on figure 2.4. The PRM is a zone of the RAM with guaranteed access management by the CPU where it will deny every external access and only allow access through the associated enclave. This region of the memory is also known as the private/protected memory or the trusted part of the application. The data managed in EPCs are mapped in plaintext, only in on-chip caches. They are encrypted and integrity-protected when they are mapped in the external (not protected) memory.

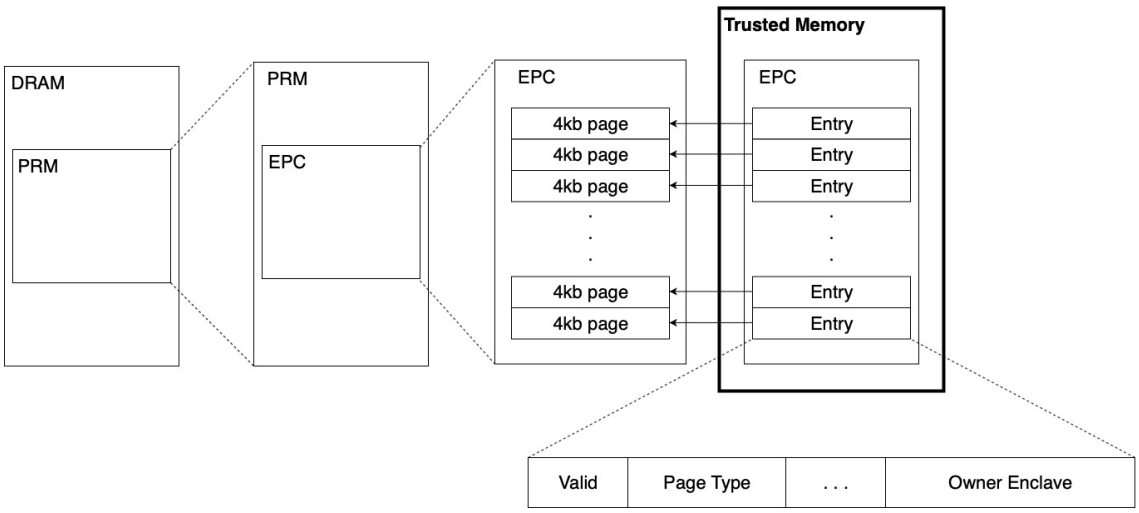


Figure 2.4: SGX Memory Architecture [37]

Because the SGX enclaves execute within the virtual address space of a process, the translation of enclave addresses must be trusted. However, since it is the OS that manages the translation between physical and virtual addresses (and the OS cannot be trusted),

SGX maintains an internal data structure called the EPCM (Enclave Page Cache Mapping) which tracks the referred mapping as well as the information described on figure 2.4 [58].

With all this information about enclave pages, the processor can now performed a controlled access management to the enclave page cache described in figure 2.5, where it will denied access not only from outside the enclave but as well as from enclaves that do not own the page of memory request creating then, an isolated memory region.

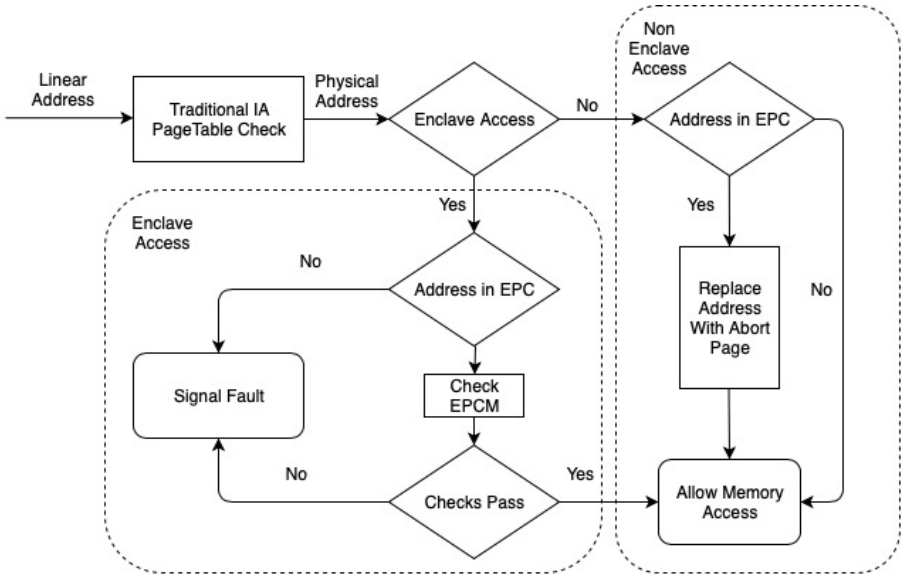


Figure 2.5: SGX Access Control [37]

The SGX will create an enclave when sensitive code needs to run by a specific SGX CPU instruction (ECREATE) and will create a unique instance of an enclave, establishing the linear address range and load the sensitive code into an EPC inside the protected memory. Once all pages are loaded into the EPC, and the loading in complete, an authentication hash is computed and is available for remote attestation so a user can verify that the code running in the enclave has not been tampered with.

An enclave must expose an API for the application to call in and advertise what services provided by the untrusted domain are needed. This is the definition of an interface boundary between the untrusted part of the code and the enclave and it is how they communicate. An Ecall is a function that the untrusted part application can call to execute some code inside the enclave and since it exposes a sensitive interface, to reduce the enclave attack surface, the number of Ecalls should be limited. On the other hand, the Ocall is a function that an enclave can call to reach a service/interface outside the enclave, on the untrusted OS. Again, calling some service out of the enclave can carry additional security risks and should be as minimal as possible [42].

When running, the execution always happens in protected mode, and to prevent data leaking, the CPU will not directly address an interrupt, fault or VM exit, but will instead emit another specific instruction (EEXIT) to properly exit the enclave, save CPU state

into the enclave and only then will service the fault.

With all this properties, Intel® through SGX set of instructions and implementation tries to achieve a secure and trusted environment with guarantees of code and data isolation, confidentiality and integrity from attackers such as the OS, hypervisor, any hardware and even physical attacks [22].

2.2.6 Sanctum

Sanctum [23] is an open-source project that shares the same as goal as Intel SGX, providing strong provable software isolation to protected the data from external hardware and software, but claims to be simpler and protect against indirect attacks called side channel attacks [46] such as cache timing attacks [20] that have been know to exist in SGX [32] [69]. These are additional software attacks that can infer private information by analysing a program's memory access patterns.

Following the minimal and simple concepts, it uses minimal invasive hardware and it does not required any modifications to the CPU major blocks, but only adds hardware to the interfaces between blocks. This allows for a respectable overhead by maintaining normal clock speeds as it does not modify the CPU core critical execution path.

Sanctum project builds on the SGX programming model and implements an architecture that deviates as little as possible from the one built by Intel. Although it differs from SGX by implementing the enclaves via a small combination of hardware extensions to RISC-V (an open source set of CPU instructions [83]) and a trusted piece of software called the security model, as SGX implements them via hardware microcode and presents a set of CPU instructions to manage the enclaves.

This security monitor is the core of the project and configures the hardware to enforce low-level rules that controls the enclaves's access policies. As explained in the Sanctum paper, "*the security monitor checks the system software's allocation decisions for correctness and commits them into the hardware's configuration registers*". One of the examples and the main points of upgrade compared to SGX is that Sanctum keeps the enclave page tables inside enclaves memory, protecting the system against the timing attacks referred above by keeping the page table dirty and accessed bits private. Their hardware extensions make sure that enclaves page tables only point to enclave memory and untrusted OS tables only point to Os memory regions and never to enclave private memory.

Sanctum is also open to the public which makes easier for security researchers to audit and find vulnerabilities and to further encourage the analysis of the code, Sanctum security monitor is written in portable C++ code and can be used across different CPU implementations.

2.2.7 ARM Trust Zone

ARM Trust Zone [11] is a technology that offers a system wide approach to security based on hardware enforced isolation built into the CPU [10]. The principle of the technology

is to separate the trusted and untrusted by two virtual processors backed by hardware access control. The two states are referred as worlds, where the first is called the secure world (SW) and the other is the normal world (NW) like figure 2.6 shows.

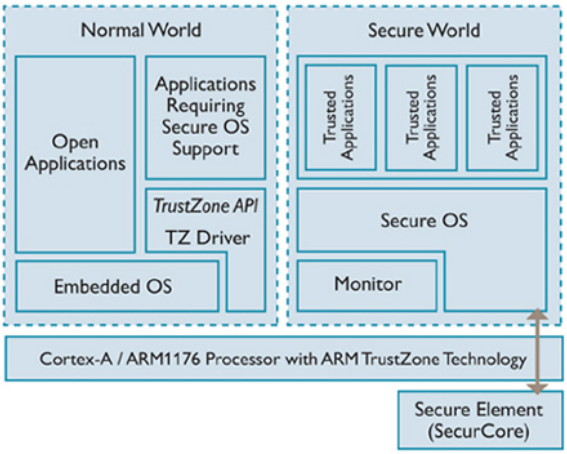


Figure 2.6: Arm TrustZone Stack [12]

The non-secure world (or normal world) is where the OS and most of the software and applications will be running, as for the secure world is where more secure and sensitive software will run and will ensures that vital information is not intercepted by a third party. The security is enforced because each of the worlds acts isolated from the other as a runtime environment with separated resources such as memory, processor, cache, controllers, interrupts. The ARM hardware has separate copies of the registers for each worlds and cross-world register access is blocked. However the Secure Monitor shown in figure 2.6 can access non secure registers while running in secure world. This means that the monitor can then implement context switching between both worlds.

When in Normal World, the application calls a specific ARM instruction call SMC (Secure Monitor Call) to call back inside the secure world and execute in code in a secure manner.

By keeping the worlds separated from each other, the ARM TrustZone can keep applications running in secure mode isolated from the normal world applications such as the OS and thus achieving another implementation of a TEE.

2.2.8 Discussion

The TCB (Trusted Computing Base) is the of set computer components (hardware), software and data that we need to trust and deem as not malicious in order to use a system. It's a group of various elements that are critical to a systems security in a way that any bug or vulnerability occurring from inside the TCB components might compromise the security and privacy of the entire system. On the other hand, security flaws and bugs from outside of the TCB should not become a security issue.

Hardware security like TPMs and TEE technologies have the goal to drastically reduce the TCB of a system, for example, remove the OS from the trusted base to make sure that a remote compromised machine vulnerability does not affect the security and privacy of a application or system.

By isolating the program from outside and uncontrollable sources like cloud infrastructures OSs, hypervisors and hardware we can more safely deploy sensitive applications to those cloud providers. That is the goal of all different implementations of trusted execution environments like the Intel SGX and the ARM TrustZone.

Although not free of some problems, SGX implementation of a TEE seems to be the most accepted technology, with serious and skilled developers and security researches always working to mitigate any vulnerability in order to create a truly trusted and isolated environment.

2.3 TEE/SGX Enabled Key Value Stores

There has been an increase trend from developers to move their applications to the cloud. It provides dynamically and almost seamlessly scaling with predict cost. Although it also means that users need to rely on the cloud providers for securing and maintaining the integrity of their applications. That means the user must trust not only the provider's staff but also its globally distributed software and hardware not to expose their private data. Today's cloud providers only aim to protect their privileged code from the untrusted code (the user's code) and do not provide any guarantees about the opposite scenario.

To mitigate this use case, and after studying and discussing the Key-Value stores technologies and also the trusted platform modules as well as the trusted execution environments, in this chapter, it will be presented how are this two topics being combined and used together.

It will be more focused on the Intel SGX platform as it is the one that will be used throughout this thesis. Currently, there are a number of databases who leverage this technology to provide a more secure environment and service. In the this chapter it's presented how they work and operate, discussed the differences between them and also the how the work planned to be performed on this thesis will solve some of the problems and caveats.

2.3.1 Trusted Execution with Intel SGX

As explained before, Intel SGX provides a trusted execution environment by running code inside the enclaves. It creates an isolated environment where we can run some instructions as securely as possible, without OS intervention.

Key-Value Stores and other database type systems can leverage this secure and isolated environment to perform queries on very sensitive data that would otherwise be vulnerable to some attacks. There are a few techniques currently implemented to use

isolated environments. Maintaining an encrypted database and using enclaves cryptographic capabilities to decrypt data and perform queries on plain text with the assurance of no data leaking is a possible use case. Also, maintaining a database fully on enclave memory, where it cannot be access by anyone other than the CPU is another way to keep the data secure by leveraging isolated and trusted execution environments. Different techniques will be furthermore discussed bellow.

As we can see, isolated and trusted execution environments are an important feature when it comes to protecting the data from the OS and Key-Values Store systems do benefit from them.

2.3.2 Circumvention of SGX Limitations

There are a few limitations and challenges of the SGX platform that we address when programming for such technology.

It starts with a big challenge of choosing and defining what parts of the program can benefit of the SGX security. As it is known, it works with two major application components, the trusted and untrusted modules of or program. The limitations have to be thoroughly analysed so we can make that definition.

The main limitations are:

- Performance
- Memory
- I/O
- syscalls

In the KVS world, as we extensively covered, performance is the major concern and there is no real way around this limitation. Using secure enclaves will definitely decrease the supposed blazing fast performance. Although, with intelligent partition between the untrusted code, which will be fast, and the trusted instructions, which will be slower we can limit the performance overhead. By separating and well defining both modules of the application, we can decrease the code that needs to run securely and find a fine compromised between security and performance. Homomorphic encryption is also a mechanism to speed up performance by performing queries directly over encrypted memory. Although, fully homomorphic encryption is not a possibility yet, so, by compromising some set of operations that are not possible, partial homomorphic encryption will speed up the performance.

With the SGX base support the access to EPC from the owner enclave is efficiently processed by hardware en/decryption logics at cache-line granularity. This means that when the cache-line is brought from EPC to the processor, it is decrypted. The hardware logic calculates the keyed hash value of the cache-line, and verifies it against the

stored hash value of the address. Internally the integrity hash values are organised in data-structures similar to Merkle Trees, to allow sub-trees to be evicted from the on-chip storage securely [36]. Due to the space and time overhead of storing and processing security metadata at fine-grained cache-line granularity for EPC, the EPC capacity is unlikely to increase significantly. For example, a huge Merkle tree for tens gigabytes of main memory at cache-line granularity will necessarily increase the integrity verification latency intolerably, as an inefficient solution that will sacrifice throughput and latency [75].

Memory sizing is also a limitation when using enclaves in SGX technology. The amount of private secure data that can be maintained by the enclave is limited to the size of the enclave cache, which is around 128 MB, being that only about 94 MB are available to the application, with the rest reserved to metadata. Now, with SGX v2 and for some operation systems, mainly Linux because of paging swap support, it can be increased up to all the memory available in the system [71] by swapping pages from the EPC to main untrusted memory, with guaranteed of confidentiality, integrity and data freshness. When evicting pages from the EPC, it is assigned a unique version number which is recorded in a new type of EPC page and the contents of the page, metadata, and EPCM information are encrypted and written out to system memory. When reloading a page back into EPC the page is decrypted and has its version and integrity checked to make sure it was not tampered with.

Although, page eviction to main untrusted memory introduces a big overhead because of encryption and decryption and integrity checks (2x - 2000x) [13]. Clever partitioning of the application into the untrusted and trusted modules will help to overcome this limitation as described in the next sections.

I/O and syscalls are limited by default on the enclave for security purposes, so it can't affect or be affected by the OS. There is a way to perform and access I/O and syscalls through the aforementioned Ecalls and Ocalls (section 2.2.5 of this thesis), but they have to be accounted for when implementing the application. To address the problem, recent proposed solutions try to reduce the frequencies of enclave exits for system calls by running threads in untrusted execution for the interaction with the operating system, communicating with the enclave thread by sharing memory. Another issue is EPC fault handling, because EPC limit requires exiting the enclave improving more the cost of paging.

2.3.3 SGX-Enabled Secure Databases

Database management service developers are now implementing secure databases ready to take advantage of Intel SGX hardware. It differs from normal databases because it runs on top of protected and encrypted memory so it can work with minimal TCB.

Next subsections present and discuss the overview of the current technology that leverages SGX to provide a secure database.

2.3.3.1 EnclaveDB

EnclaveDB [61] is a privacy enhanced and secure database that works alongside with Intel SGX and provides a Structured Query Language (SQL). It uses its technology to maintain all sensitive information inside SGX enclaves in order to keep them secure from a threat model of strong adversaries that can control the entire software stack on the database server. It resists attack from the administrator server, the database administrator and attacker who may compromised the operating system, the hypervisor or the database server.

Following Intel's application guidelines, EnclaveDB has a two part architecture: trusted (running on the enclave) and untrusted modules. The enclave hosts a query processing engine, natively compiled stored procedures and a trusted kernel which provides API's for sealing and remote attestation. The untrusted host process runs all other components of the database server. Figure 2.7 shows the architecture of the enclaveDB server-side.

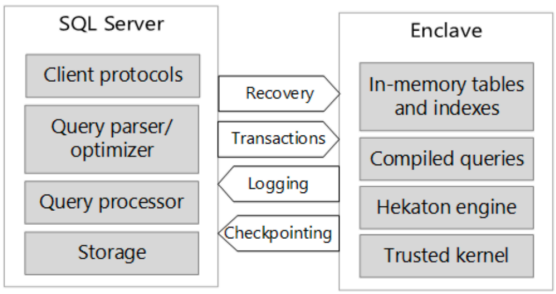


Figure 2.7: Server-side components of EnclaveDB

Leveraging TEE, EnclaveDB then provides a database with a SQL interface and guarantees confidentiality and integrity with low overhead. With its design it also reduces the TCB to a smaller set than any other "normal"database.

2.3.3.2 Pesos DB

Pesos [49] is a secure implementation of object storage services like Amazon S3 [7], Azure Blob Storage [18], Google Cloud Storage [31] among others. In these current large-scale services, due to their complexity, the risk of confidentiality and integrity violations increase significantly. This storage systems are characterised by multiple layers of software and hardware stacked together which means the access policies for ensuring confidentiality and integrity are scattered across different code paths and configurations, thus exposing the data to more security vulnerabilities. Furthermore, untrusted third-party cloud platforms expose an additional risk of unauthorised data access by a malicious administrator.

Pesos allows clients to specify per-object security policies concisely and separately from the remaining storage stack. It also provides cryptographic attestation for the stored objects and their associated policies to verify the policy enforcement.

It enforces this policies by leveraging the Intel SGX for trusted execution environments and Kinetic Object Storage [48] for trusted storage (secure storage - not the focus of this thesis). It structures a policy-compiler, its binary-format interpreter, per-object policy metadata, and the enforcement logic into a single layer of the storage stack. With this unification, it drastically reduces the TCB when compared to the order cloud services. Then it uses the trusted execution environment provided by SGX to connect directly Kinetic disk through an encrypted Ethernet connection allowing for object transfer and policy enforcement securely without any intermediate layers in the storage stack.

2.3.3.3 Speicher

Speicher [19] is a secure LSM-based Key-Value store that uses Intel SGX and it ensures not only strong confidentiality and integrity properties, but also data freshness to protect against rollback/forking attacks. It leverages SGX technology to achieve those security characteristics focusing on providing a **persistent** service, tolerant to system faults and securely recovering from crashes. It also tackles in interesting ways, two of the major limitations of SGX: Memory Limits and Performance.

Implementing a Key-Value Store has a major requirement - High performance and low latency queries for big data structures. As already discussed, SGX has some memory limits. The enclave memory is located in the Enclave Page Cache (EPC) which is limited to 128 MB with about 94 MB available for application use (the rest being reserved for metadata). To allow creation of enclaves with bigger size than EPC, the OS can use secure paging mechanism where it evicts pages to untrusted memory. Although with page encryption, decryption and integrity checks, this solution introduces high overheads (2× - 2000×) [13].

To address this performance and memory problems, the developers of Speicher implemented the following custom features (from Speicher public paper):

- *"**I/O library for shielded execution**: Direct I/O library for shielded execution. The I/O library performs the I/O operations without exiting the secure enclave; thus it avoids expensive system calls on the data path."*
- *"**Asynchronous trusted monotonic counter**: Trusted counters to ensure data freshness. The counters leverage the lag in the sync operations in modern KVS to asynchronously update the counters. Thus, they overcome the limitations of the native SGX counters."*
- *"**Secure LSM data structure**: Secure LSM data structure that resides outside of the enclave memory while ensuring the integrity, confidentiality and freshness of the data. Thus, the LSM data structure overcomes the memory and I/O limitations of Intel SGX."*

The technology leverages SGX with a clever partition between trusted and untrusted modules of the application. By maintaining the encrypted data on untrusted memory

hardware it addresses the memory and persistent limitations, and by keeping some information in secure enclave memory and with a good I/O library it overcomes (to an extent) the performance issues.

2.3.3.4 ShieldStore

ShieldStore [47] is a *"(...) shielded in-memory Key-Value Storage with SGX"*. It aims to provide a very fast and low latency queries over very large data trying to overcome the SGX memory limitation. It accomplice's it by maintaining the majority of the data structures in the non-enclave memory region, addressing as well the performance issue by not relaying on the page-oriented enclave memory extension provided by SGX.

ShieldStore runs server-side in the enclave to protect encryption keys and for remote attestation and it is used to perform all the KVS logic. It uses a hashed index structured but places it in the unprotected memory region instead of the enclave EPC. As the main data structure is not protected by the SGX hardware, each data entry must be encrypted by ShieldStore in the enclave, and written to the main hash table.

The main flow and architecture is as described on figure 2.8.

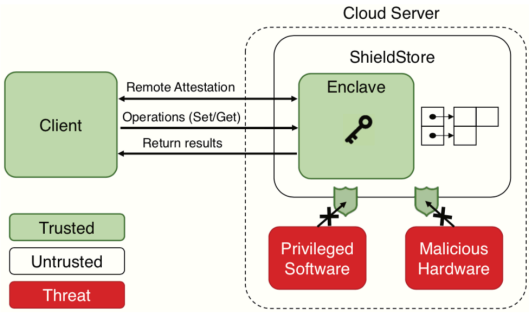


Figure 2.8: Overview of ShieldStore

First the client remote attests the server-side (1) verifying SGX support of the processor, the code, and other critical memory state of an enclave. In a second step, the client and the server exchange sessions keys (2) in order to establish a secure connection, using Intel SGX libraries to do so. Using this newly generated session key, the client sends a request for an operation (3). The server deciphers and verifies the request and accesses the Key-Value Store (4). Clients do not access the server-side ciphertexts neither need to know the encryption key used by the server to encrypt the values. The server will then decrypt the data from the storage, encrypted it again with the session key and reply to the client (5). All accesses to the KVS have integrity checks.

2.3.4 Discussion

Concluding, EnclaveDB (section 2.3.3.1) and Pesos (section 2.3.3.2) presents secure databases and objects storage systems respectively, using SGX, but EnclaveDB assumes that SGX

supports large enclaves whose size is an order of several hundred GBs and Pesos restricts the size of data structure to the size of EPC. On the other hand, Speicher (section 2.3.3.3) and ShieldStore (section 2.3.3.4) proposes a store that alleviates the memory limitation of Intel SGX by storing encrypted data on untrusted memory regions. Speicher and ShieldStore have similar architectures, but the former is primarily design for persistence storage and the latter is focused on a fast in-memory key-value store.

We can now conclude that the clever partitioning of the application into trusted and untrusted parts is really important when programming with Intel SGX. It directly affects syscalls and I/O, performance and memory of the service.

The long goal of this thesis is to implement a system with characteristics from the databases present above. In terms of performance we plan to implement partial homomorphic encryption, so it allows to perform operations directly over in-memory encrypted data. This will be a challenge, as fully homomorphic encryption is not yet practical [30], so adaptations must be made, but performance increase are expected over the databases presented above, by not needing to decrypt the data in secure execution. Persistence will also be a requirement just like some of the databases presented.

For trusted execution with SGX, extensive research is needed to partition the application in the two necessary modes to circumvent persistence, performance and memory SGX limitations. It will also be researched and tested the ability to provide built-in replication and availability with SGX.

2.4 Related Work Balance and Critical Analysis

In the current days, computer scientists are always looking for a secure, fast and cheap environment to develop applications. As we know, it is not feasible to have all three of this elements working flawlessly without any compromises. Although, by combining in-memory key value stores, trusted remote execution environments and cloud providers, developers can now have a practical example of what would be to develop for a privacy enhanced secure system with reduce costs by using cloud providers and with better reassurances that the hardware and software that it's out of the control of the user will have a minimal impact on sensitive data and code of an application. By adding the performance benefits of an in-memory key value store and all of its technology, like built-in security, built-in replication and persistence we can in the best of our abilities today, combine the best of the three worlds without compromising too much on any of them.

In this thesis we will compare different kinds of approaches to implement a fast system with the assurance of a secure data flow that can easily be deployed into the cloud without fear of any components out of our control.

Chapter 3

Approach to Elaboration Phase

This chapter presents an overview of the planned elaboration phase. In section 3.1 the objectives and contributions are more refined and matched against the related work discussed in chapter 2, also discussing the scenario and environment where the system should fall. Section 3.2 presents a reference of the system model and blueprint architecture of the targeted solution. The architecture and implementation guidelines of the solution are approached in section 3.3.

After exposing the planned implementation and architecture, section 3.4 explains how and where it is planned to test the system, whereas section 3.5 will complement it with what metrics will be gathered and how they will be compared with each other.

3.1 Refinement of Objectives and Contributions

The goal of this dissertation as explained on chapter 1, is the design, development and validation with experimental evaluation of a secure in-memory storage (based on a "key-value"model), supported by a hardware-enabled trust computing base.

Regarding the security assumptions, the solution will provide: (i) hardware-isolated in-memory processing engine, designed as a hardware-isolated container (docker) facility, enclaved within the Intel-SGX protection guarantees; (ii) hardware-isolated communication endpoints for client access, providing TLS tunnelling with strong TLS 1.3 endpoint encryption parameterisations and support for mutual client/server authentication, and (iii) privacy-enhanced operations to be directly processed on encrypted data sets in memory. The former facility is particularly interesting to combine the possibility to manage protected memory for small data sets and also searchable encrypted data sets that are far larger than the protected memory limits imposed by the SGX memory mapping facility. Furthermore, the solution will target main data structures commonly use

fine-grained data items that can include pointers, complex composite types and keys, which do not match well with the coarse-grained paging of the SGX memory extension technique.

In the next subsections we will align the implementation ideas starting by refining the circumvention of limitations in SGX and the threat model assumptions to address our solution.

3.1.1 SGX Limitations Refinement

Section 2.3.2 describes in detail the limitations of SGX. In our approach we will intend to design a solution that can be leveraged from conventional reference KVS technology, giving the possibility to manage small datasets but also larger datasets (directly mapped in non-protected memory). To circumvent the problem our solution must combine the possibility to use the internal capabilities native to SGX with the possibility of supporting operations managing datasets encrypted in memory, with such operations executed directly over encrypted data. This facility will be provided by the use of partial homomorphic encryption constructions, with data initially encrypted and submitted to the key-value-store solution with cryptographic keys only managed in the client side. Our solution will be designed in order to be possible the support for fine-grained key-value encryption, driven form the application requirements. Our target is the support of a variety of operations provided in a typical KVS API, taking REDIS as the reference solution and in order to support a considerable number of queries currently used by many REDIS-supported applications.

3.1.2 Adversary Model

As the baseline, our threat model will lie on the protection overview stated in SGX's paper [51]: *SGX prevents all other software from accessing the code and data located inside an enclave including system software and access from other enclaves. Attempts to modify an enclave's contents are detected and either prevented or execution is aborted*, which falls in the following adversary model:

Isolation by trusted containerization from malicious code: The system performs and protects its data from an attacker capable of compromising the system through another application installed on the same system or malicious code existent or injected in the OS or OS hypervisor layers;

Privacy protection against insider "Honest but Curious"System Administrators: The system must be able to protect from an attacker with root access to the machine, with permissions to access and monitor memory-mapped data. This is relevant because we will target our solution as a candidate solution to protect data privacy in a cloud-based key value store solution as a service, preventing data-leakage vulnerabilities exploitable by insider incorrect users or system-administrators.

Network Attacks: All communication to and from the system (supporting client-service operations) should be secure, using proper strong cryptographic parameterisations for TLS 1.3, mutual authenticated handshakes and TLS endpoint executions isolated in SGX-enabled TLS tunnels in communication containers, avoiding attacks against the authentication of the service endpoints, as well as, attacks against the integrity and confidentiality of data flows supporting REST/HTTPS operations.

File system and memory access attacks: All sensitive data residing outside protected memory should be encrypted and operated in the encrypted form. An attacker can access the physical disks and hardware without the sensitive data being exposed.

3.1.3 Other System Assumptions

With the above security baseline considered for the threat model assumptions, the solution must be resilient to malicious privileged attacks and certain physical attacks. With a controlled and reduced hardware-shielded trust computing base, we want to design a solution that does not rely on the security of operating system managed by cloud providers. Furthermore, our solution must be also resilient to direct conventional physical attacks, such as cold boot attacks, which attempt to retain the DRAM data by freezing the memory chip or even bus probing to sense and to read exposed memory channel between the processor and memory chips. The only weaknesses not covered in our concerns with be the SGX lack of protection for side-channel attacks.

The system planned has certain assumptions and aspects that are considered to be out of scope for this dissertation:

- **Trusted Client** - The client side is assumed to be completely trusted and correct.
- **DoS and DDoS** attacks are out of scope.
- **Side Channel Attacks** - It is out of scope any side channel attacks or any related attack not present in SGX's threat model.
- **Physical and Hardware attacks** exploring the SGX processing model and its isolation guarantees are out of scope, namely those presented above initially addressed in chapter 2, section 2.2.

3.2 System Model Approach

The main goal of this project is modelled in figure 3.1. As shown, the model can be divided in three main components - the **client** which was already defined as trustable, the **communications** and the **backend service**.

The system should comply with the adversary and threat model explained in section 3.1.2. The user should not be aware of implementation details and should have seamless interaction with the system despite the architecture and implementation provided by the

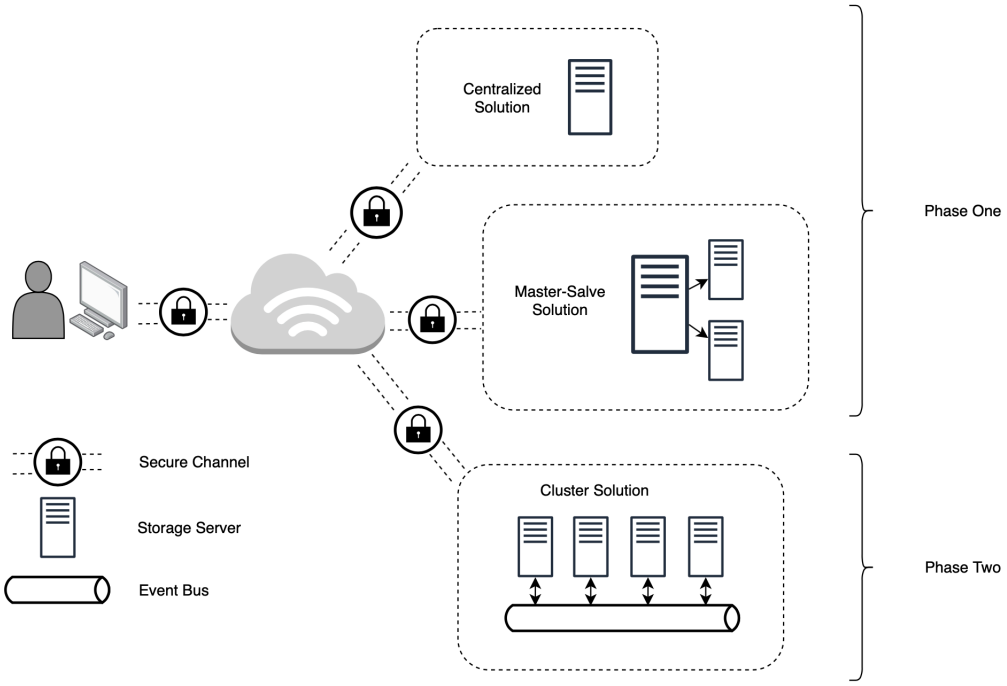


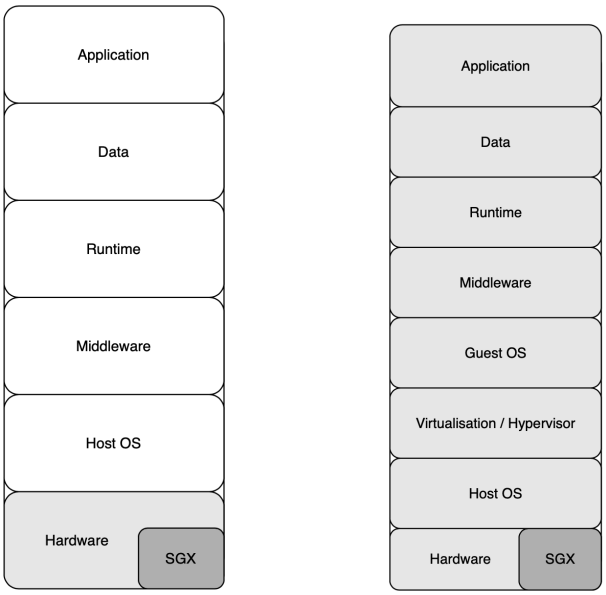
Figure 3.1: System Model Details

backend, meaning that all solution should expose the same API and support an interface as equal as possible to the unprotected version of the Key-Value Store. We must mention that the above system model will address the replication facilities as leveraged from the REDIS solution and trying to offer the same availability conditions as the original REDID cluster service. This availability provided by the replication of REDIS instances that can run as cloud service instances (that can be distributed among different cloud providers and geo-replicated datacenters) only offers consistency guarantees under a fail-stop model (not extended to byazntine-fault-tolerance or byzantine intrusion tolerance, that are conditions out of scope of our planned dissertation).

3.3 Planned Architecture and Implementation

Cloud providers offer their services and applications in different stacks and computing infrastructures. They can be categorised in two main and most common: IaaS (Infrastructure as a Service), and SaaS (Software as a Service). These are terms to represent how much of the stack is available for user customisation and how much is managed by the provider. Figure 3.2 shows the two stacks that where the shaded components represent the ones managed by the provider.

We will start development with a stack like figure 3.2a as it is the simpler to customise and change to our needs especially for development. Everything besides the physical hardware can be customisable and it is managed by the user. In the end, we would like to expose the solution as a SaaS, like figure 3.2b where it provides to the user a more



(a) IaaS Stack (b) PaaS Stack

Figure 3.2: Computing Stacks

abstracted environment, requiring less knowledge of computing bases and setups [82].

As shown on figure 3.2 and to achieve the objectives and goals of the project (1.3), the hardware chosen was Intel's SGX secure module. As explained in section 2.2.5, this technology has the means and the reach to provide a truly isolated environment has required in this dissertation. The stack will be running a Linux Ubuntu Server [68] operating system as the Host OS, version 18.04 LTS. This version has compatibility with Intel's SGX hardware, with the help of its SDK and drivers [43, 50] and with some software needed in another layer of the stack.

Redis (v5.0.7 [63]) will be the key-value store application that will be worked on. This is the most used and most known KVS technology, written in C language and will be deployed on the application component of the stack. To support large datasets, given SGX limitations explained in section 2.3.2, all data will be kept outside the enclave and will always be encrypted. The client can then decide to perform queries in encrypted from directly over the encrypted memory, sparing encryption and decryption cycles on the server. This will speed up the performance and will keep encryption keys away from the server in the cloud. Keys would be managed by the client which is considered trustable by the scope of this dissertation. Although this technique should help performance, it also means that system can lose some abilities and a set of operations. Fully homomorphic encryption is not feasible yet, so this solution will not be a full fledged solution. To this end, the solution will use the HomoLib, which is a JAVA Partial Homomorphic Cryptographic Library developed be the NOVA LINCS Research Center [74].

To help with SGX integration a middleware software called Graphene [34, 81] will be used. This technology is a library OS that *"with SGX support (...) can secure a critical*

application in a hardware-encrypted memory region. Graphene can protect applications from a malicious system stack with minimal porting effort". This library has already been tested with Linux Ubuntu Server 18.04 running as the OS.

Communications will be secure with the help of SGXStunnel [73], a prototype proxy to support TLS tunnelling with endpoints executed in a trusted execution environment provided by Intel SGX. SGX termination means that at no time the data exchanged between the client and the server is exposed in clear text to any outsider because the packets are decrypted inside the secure module, which allows the network drivers, modules and physical cards to be removed from the TCB.

Services and runtime environments will be coded using Java (Java SE 11 (LTS)) and the deployment of all softwares and services will be done with containers using docker technology [28]. Graphene already supplies integration with docker [35].

3.4 Planned Testbench Environments

The services will be deployed and tested in two different environments. The **development environment** translates to a local virtual machine that aims to simulate and be as close as possible to the production environment, but providing ease of development and rapid deployments. The **production environment** corresponds to a cloud provider that offers SGX dedicated hardware, like OVHcloud [60]. The machines used should have the following specs:

Listing 3.1: Machine Specifications

```
Dedicated Server Node
Processor: Intel 2x Xeon Silver 4214 - 24c/48t - 22.GHZ/3.2Ghz
Memory: 192 GB
Hard Drive: NVMe, SATA available
Public Network: Beginning at 1 Gbps
Private Network: Beginning at 2 Gbps
CloudLinux (Ubuntu 18.4 LTS Server 64 bits)
```

The tests will pe performed by a combination of the built-in Redis-Client benchmark tests [39] and to eliminate any bias claims, an external tester like the Yahoo! Cloud Serving Benchmark [85].

3.5 Relevant Evaluation Criteria

The testers and benchmark clients will evaluate metrics that can be compared with a non secure solution. On the network layer, latency and throughput (as operations per seconds (ops/s)) and on the server side we will monitor the resources of the machines, like memory consumption, CPU load and also power consumption during the performance tests. Resource monitoring metrics should be gathered by the built in Linux tools like *htop*, *lsof* or *vmstat*, and/or docker built-in tools like *docker stats*.

Chapter 4

Elaboration Plan

This chapter proposes a work plan for the elaboration phase of this dissertation. First, in section 4.1, we summarise the planned tasks as well as their planned time frames. On the annexes it is presented a Gantt chart and a table better detailing the tasks, as well as referring the relevant related dependencies and related milestones.

4.1 Work Plan

Our work plan for the elaboration phase will be conducted during the period from 2/Mar/2020 to 15/Sep/2020, summarised as follow:

- **2-13/Mar**: System model and architecture refinements and implementation specifications.
- **9/Mar-8/Apr**: Setup of the development environment and implementation and test of a pilot-prototype for a SGX docker-based small-scale and single REDIS instance.
- **13/Apr-22/May**: Development of the planned solution (first prototype).
- **4/May-29/May**: Functional, architectural and initial performance evaluation tests with the first prototype.
- **1/Jun-3/Jul**: Development of the planned solution (second prototype, including a replicated solution full-compliant with the primary-backup and cluster model as provided by the original REDIS solution).
- **22/Jun-10/Jul**: Functional, architectural and initial performance evaluation tests with the first prototype.

- **13/Jul-25/Jul**: Deployment of the final Prototype as a Cloud-Based Solution as a Service (running in OVH dedicated instances and datacenters).
- **22/Jul-8/Aug** and **22/Aug-18/Sep**: Final tests in the Cloud Prototype.
- **30/Mar-18/Sep**: Thesis report writing (in distributed cycles addressing different chapters and the final review phase planned for 11-18 Sep).

Figure II.1 details the tasks shown in the work plan Gantt Chart (figure I.1). For each task a description is available to better explain the work planned for the task, and a dependency to another task if relevant.

There are 10 main categories, called milestones or epics, that were identified: **System Specification**, **Development Environment**, **Production Environment**, **Prototype Setup**, **Prototype**, **Prototype Communication**, **Experimental Evaluation**, **Prototype Publication**, **Dissertation Writing** and **Dissertation Presentation**. Each task will be appointed to epic task.

Bibliography

[1] *83% Of Enterprise Workloads Will Be In The Cloud By 2020*. Accessed: 2020-01-11. URL: https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020/#2e513c5b6261.

[2] *A brief history of cloud computing*. Accessed: 2020-01-11. URL: https://www.ibm.com/blogs/cloud-computing/2014/03/18/a-brief-history-of-cloud-computing-3/.

[3] *A history of cloud computing*. Accessed: 2020-01-11. URL: https://www.computerweekly.com/feature/A-history-of-cloud-computing.

[4] *Aerospike*. Accessed: 2019-06-16. URL: https://www.aerospike.com.

[5] *Aerospike*. Accessed: 2019-06-16. URL: https://www.aerospike.com/docs/guide/security/index.html.

[6] *Amazon Dynamo DB*. Accessed: 2019-06-16. URL: aws.amazon.com/dynamodb.

[7] *Amazon S3 - Cloud Storage*. Accessed: 2019-07-09. URL: https://aws.amazon.com/s3/.

[8] *Amazon Web Services*. Accessed: 2019-06-16. URL: https://aws.amazon.com.

[9] *Apple's iCloud service suffers cyber-attack in China, putting passwords in peril*. Accessed: 2020-01-11. URL: https://www.washingtonpost.com/news/the-switch/wp/2014/10/21/apples-icloud-service-suffers-cyber-attack-in-china-putting-passwords-in-peril/.

[10] *ARM TrustZone*. Accessed: 2020-01-08. URL: https://developer.arm.com/ip-products/security-ip/trustzone.

[11] *ARM TrustZone Security Whitepaper*. Tech. rep. PRD29-GENC-009492C. ARM Limited, Dec. 2008. URL: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[12] *ARM TrustZone Stack Image*. Accessed: 2020-01-08. URL: https://malware.news/t/introduction-to-trusted-execution-environment-arms-trustzone/20823.

[13] 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.

[14] *Around 75% of Open Redis Servers Are Infected With Malware*. Accessed: 2020-02-18. URL: https://www.bleepingcomputer.com/news/security/around-75-percent-of-open-redis-servers-are-infected-with-malware/.

[15] *Attestation and Trusted Computing - CSEP 590: Practical Aspects of Modern Cryptography*. Accessed: 2019-11-26. URL: https://courses.cs.washington.edu/courses/csep590/06wi/finalprojects/bare.pdf.

[16] *Attestation Identity Key (AIK) Certificate Enrollment Specification*. Accessed: 2019-11-26. URL: https://www.trustedcomputinggroup.org/wp-content/uploads/IWG-AIK-CMC-enrollment-FAQ.pdf.

[17] *AWS hit by major DDoS attack*. Accessed: 2020-01-11. URL: https://www.techradar.com/news/aws-hit-by-major-ddos-attack.

[18] *Azure Blob Storage*. Accessed: 2019-07-09. URL: https://azure.microsoft.com/en-us/services/storage/blobs/.

[19] M. Bailleu, J. Thalheim, P. Bhatotia, C. Fetzer, M. Honda, and K. Vaswani. "SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution." In: *17th USENIX Conference on File and Storage Technologies (FAST 19)*. Boston, MA: USENIX Association, 2019, pp. 173–190. ISBN: 978-1-931971-48-5. URL: https://www.usenix.org/conference/fast19/presentation/bailleu.

[20] S. Banescu. *Cache Timing Attacks*. July 2011. URL: https://www.academia.edu/3224323/Cache_Timing_Attacks.

[21] *Benefits of Hardware Trusted Modules*. Accessed: 2019-11-30. URL: https://www.hardware-security-module.com/benefits/.

[22] V. Costan and S. Devadas. *Intel SGX Explained*. Cryptology ePrint Archive, Report 2016/086. 'https://eprint.iacr.org/2016/086'. 2016.

[23] V. Costan, I. Lebedev, and S. Devadas. *Sanctum: Minimal Hardware Extensions for Strong Software Isolation*. Cryptology ePrint Archive, Report 2015/564. 'https://eprint.iacr.org/2015 2015.

[24] *Data leaks: The most common sources*. Accessed: 2020-02-18. URL: https://www.zdnet.com/pictures/data-leaks-the-most-common-sources/13/.

[25] *Data leaks: The most common sources*. Accessed: 2020-02-18. URL: https://www.zdnet.com/pictures/data-leaks-the-most-common-sources/12/.

[26] *Data leaks: The most common sources*. Accessed: 2020-02-18. URL: https://www.zdnet.com/pictures/data-leaks-the-most-common-sources/14/.

[27] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. "Dynamo: Amazon's Highly Available Key-value Store." In: *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*. SOSP '07. Stevenson, Washington, USA: ACM, 2007, pp. 205–220. ISBN: 978-1-59593-591-5. DOI: 10.1145/1294261.1294281. URL: http://doi.acm.org/10.1145/1294261.1294281.

[28] *docker*. Accessed: 2020-02-18. URL: https://www.docker.com.

[29] *Document Store*. Accessed: 2019-06-16. URL: https://aws.amazon.com/nosql/document/.

[30] C. Gentry, S. Halevi, and N. P. Smart. "Homomorphic Evaluation of the AES Circuit." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 850–867. DOI: 10.1007/978-3-642-32009-5_49. URL: https://doi.org/10.1007/978-3-642-32009-5_49.

[31] *Google Cloud Storage*. Accessed: 2019-07-09. URL: https://cloud.google.com/storage/.

[32] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller. "Cache Attacks on Intel 'SGX'." In: *Proceedings of the 10th European Workshop on Systems Security - 'EuroSec' '17*. 'ACM' Press, 2017. DOI: 10.1145/3065913.3065915. URL: https://doi.org/10.1145/3065913.3065915.

[33] *Graph DBMS*. Accessed: 2019-06-16. URL: https://aws.amazon.com/nosql/graph/.

[34] *Graphene Library OS with Intel SGX Support*. Accessed: 2020-02-16. URL: https://github.com/oscarlab/graphene.

[35] *Graphene-SGX Secure Container (GSC): Automatic Protection of Containerized Applications with Intel SGX*. Accessed: 2020-02-18. URL: https://github.com/rainfld/gsc.

[36] S. Gueron. *A Memory Encryption Engine Suitable for General Purpose Processors*. Cryptology ePrint Archive, Report 2016/204. 2016. URL: https://eprint.iacr.org/2016/204.

[37] S. K. Haider, H. Omar, M. Ahmad, C. Jin, and M. van Dijk. *Intel's SGX In-depth Architecture*. URL: https://scl.engr.uconn.edu/courses/ece6095/lectures/sgx_architecture.pdf.

[38] *Hardware Trusted Modules*. Accessed: 2019-11-30. URL: https://resources.infosecinstitute.com/tpms-or-hsms-and-their-role-in-full-disk-encryption-fde/.

[39] *How fast is Redis?* Accessed: 2020-02-16. URL: https://redis.io/topics/benchmarks.

[40] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar. *Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud*. Cryptology ePrint Archive, Report 2015/898. 2015. URL: https://eprint.iacr.org/2015/898.

[41] *Intel SGX*. Accessed: 2020-01-04. URL: https://software.intel.com/en-us/sgx.

[42] *Intel® Software Guard Extensions (Intel® SGX) - Developer Guide*. Accessed: 2020-01-25. URL: https://download.01.org/intel-sgx/linux-2.2/docs/Intel_SGX_Developer_Guide.pdf.

[43] *Intel® Software Guard Extensions SDK for Linux*. Accessed: 2020-02-18. URL: https://01.org/intel-softwareguard-extensions.

[44] S. IT. *Key-Value Stores*. Accessed: 2019-06-16. URL: https://db-engines.com/en/article/Key-value+Stores.

[45] S. IT. *Key-Value Stores Ranking*. Accessed: 2019-06-16. URL: https://db-engines.com/en/ranking/key-value+store.

[46] A. K. Khan and H. J. Mahanta. "Side channel attacks and their mitigation techniques." In: *2014 First International Conference on Automation, Control, Energy and Systems ('ACES')*. IEEE, Feb. 2014. DOI: 10.1109/aces.2014.6807983. URL: https://doi.org/10.1109/aces.2014.6807983.

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

[48] *Kinetic Object Storage*. Accessed: 2019-07-09. URL: https://storageioblog.com/seagate-kinetic-cloud-object-storage-io-platform/.

[49] R. Krahn, B. Trach, A. Vahldiek-Oberwagner, T. Knauth, P. Bhatotia, and C. Fetzer. "Pesos." In: *Proceedings of the Thirteenth EuroSys Conference on - EuroSys 18*. ACM Press, 2018. DOI: 10.1145/3190508.3190518. URL: https://doi.org/10.1145/3190508.3190518.

[50] *linux-sgx-driver*. Accessed: 2020-02-18. URL: https://github.com/intel/linux-sgx-driver.

[51] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue, and U. Savagaonkar. *Innovative Instructions and Software Model for Isolated Execution*. Tech. rep. Intel Corporation, Aug. 2013. URL: https://software.intel.com/sites/default/files/article/413936/hasp-2013-innovative-instructions-and-software-model-for-isolated-execution.pdf.

[52] *Memcached*. Accessed: 2019-06-16. URL: http://www.memcached.org.

[53] *Memcached Github*. Accessed: 2019-06-16. URL: https://github.com/memcached/memcached/wiki/Overview.

[54] *Microsoft Azure Cache For Redis*. Accessed: 2019-06-16. URL: https://azure.microsoft.com/en-us/services/cache/.

[55] *Microsoft Azure Cosmos DB*. Accessed: 2019-06-16. URL: https://azure.microsoft.com/en-us/services/cosmos-db/.

[56] *Microsoft Azure Services*. Accessed: 2019-06-16. URL: https://azure.microsoft.com/en-us.

[57] *Microsoft Security Intelligence Report Volume 22*. Accessed: 2020-01-11. URL: https://www.microsoft.com/security/blog/2017/08/17/microsoft-security-intelligence-report-volume-22-is-now-available/.

[58] M. Minkin. "Improving Performance and Security of Intel SGX." Master's thesis. Israel Institute of Technology, Dec. 2018.

[59] *Open Redis Servers Infected with Malware*. Accessed: 2020-02-18. URL: https://www.infosecurity-magazine.com/news/open-redis-servers-infected-with/.

[60] *OVHcloud - Intel Software Guard Extensions (SGX)*. Accessed: 2020-02-16. URL: https://www.ovh.ie/dedicated_servers/software-guard-extensions/.

[61] C. Priebe, K. Vaswani, and M. Costa. "EnclaveDB – A Secure Database using SGX." In: *To appear in the Proceedings of the IEEE Symposium on Security & Privacy, May 2018*. IEEE, May 2018. URL: https://www.microsoft.com/en-us/research/publication/enclavedb-a-secure-database-using-sgx/.

[62] *Professor John McCarthy*. Accessed: 2020-01-11. URL: https://cs.stanford.edu/memoriam/professor-john-mccarthy.

[63] *Redis*. Accessed: 2019-06-16. URL: https://redis.io.

[64] *Redis Multi Model Store*. Accessed: 2019-06-16. URL: https://db-engines.com/en/system/Redis.

[65] *Redis Performance Benchmark*. Accessed: 2019-06-16. URL: https://redislabs.com/docs/nosql-performance-benchmark/.

[66] *Redis Security*. Accessed: 2019-06-16. URL: https://redis.io/topics/security.

[67] *SASL Rfc*. Accessed: 2019-06-16. URL: https://tools.ietf.org/html/rfc2222.

[68] *Scale out with Ubuntu Server*. Accessed: 2020-02-16. URL: https://ubuntu.com/server.

[69] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. "Malware Guard Extension: Using SGX to Conceal Cache Attacks." In: *arXiv e-prints*, arXiv:1702.08719 (Feb. 2017), arXiv:1702.08719. arXiv: 1702.08719 [cs.CR].

[70] *Search Engine Database*. Accessed: 2019-06-16. URL: https://aws.amazon.com/nosql/search/.

[71] *Sgx Memory Limits*. Accessed: 2019-06-16. URL: https://software.intel.com/en-us/forums/intel-software-guard-extensions-intel-sgx/topic/670322.

[72] *SGX Secure Enclaves in Practice: Security and Crypto Review*. Black Hat USA 2016 Security Conference. URL: https://www.youtube.com/watch?v=0ZVFy4Qsryc.

[73] *SGXSTUNNEL - SGX ENABLED TLS AND SSH TUNNEL*. Accessed: 2020-02-16. URL: http://nova-lincs.di.fct.unl.pt/prototype/251.

[74] *SJHOMOLIB*. Accessed: 2020-02-18. URL: http://nova-lincs.di.fct.unl.pt/prototype/233.

[75] M. Taassori, A. Shafiee, and R. Balasubramonian. "VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures." In: *ASPLOS '18*. 2018.

[76] *Tesla cloud systems exploited by hackers to mine cryptocurrency*. Accessed: 2020-02-18. URL: https://www.zdnet.com/article/tesla-systems-used-by-hackers-to-mine-cryptocurrency/.

[77] *The Benefits Of Moving To The Cloud*. Accessed: 2020-01-11. URL: https://www.forbes.com/sites/forbestechcouncil/2017/05/19/the-benefits-of-moving-to-the-cloud.

[78] *The Cambridge Analytica scandal changed the world – but it didn't change Facebook*. Accessed: 2020-02-18. URL: https://www.theguardian.com/technology/2019/mar/17/the-cambridge-analytica-scandal-changed-the-world-but-it-didnt-change-facebook.

[79] *Time Series Databases*. Accessed: 2019-06-16. URL: https://www.forbes.com/sites/metabrown/2018/03/31/get-the-basics-on-nosql-databases-time-series-databases/.

[80] *Trusted Platform Modules*. Accessed: 2019-07-09. URL: https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/trusted-platform-module-overview.

[81] 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)*. URL: https://www.usenix.org/system/files/conference/atc17/atc17-tsai.pdf.

[82] *Understanding the cloud computing stack: SaaS, PaaS, IaaS*. Accessed: 2020-02-16. URL: https://support.rackspace.com/how-to/understanding-the-cloud-computing-stack-saas-paas-iaas/.

[83] A. Waterman, Y. Lee, D. A. Patterson, K. Asanovic, V. I. U. level Isa, A. Waterman, Y. Lee, and D. Patterson. *The RISC-V Instruction Set Manual*. 2014.

[84] *What is a Trusted Execution Environment (TEE)?* Accessed: 2019-12-03. URL: https://www.trustonic.com/news/technology/what-is-a-trusted-execution-environment-tee/.

[85] *Yahoo! Cloud Serving Benchmark*. Accessed: 2020-02-16. URL: https://github.com/brianfrankcooper/YCSB.

ANNEX I

Work Plan Gantt

Figure I.1 presents the tasks explained in section 4.1 destributed along time in a Gantt chart.

Figure I.1: Work Plan

ANNEX II

Work Plan Details

Figure II.1 details the tasks shown in the work plan Gantt Chart (I.1). For each task a description is available to better explain the work planned for the task, and a dependency to another task if relevant

#	Task	Description	Dependencies	Milestones
1	**System model and architecture refinement**	Refinement of the specified system model and architecture	-	System Specification
1.1	Specification of adversary model and topology	Refinement of the specified adversary model and topology	-	System Specification
1.2	Redefinement of system assumptions	Refinement of the specified system assumptions	1.1	System Specification
2	**Setup and configuration of development enviroment**	Setup and configure the development environment with all the necessary hardware and software components, creating a suitable development and testing environement	1	Development Environemnt
2.1	Setup and Configuration - OS and Software necessary	Setup VM with suitable operating system and all necessary software	1.1	Development Environemnt
2.2	Setup and Test of virtualization helper - Graphene	Installations and test of the virtualization helper to better integrate with the SGX hardware. Testing with a small test application to make sure everything is working together	2.1	Development Environemnt
2.3	Preparation of environment for the diferent phases	Prepare the development environment to receive the 2 different phases and different evaluations mechanisms isolated from each other	2.1	Development Environemnt
2.4	Component revision and integration testing	Revise and test integration between all components	2.2, 2.3	Development Environemnt
3	**Development and Prototype Implementation - Phase 1**	Prototype first phase development	2	Prototype
3.1	Preparation of the stack and software for phase 1	Final adaptation of the developement environment stack to work with the prototype phase one	2	Prototype Setup
3.2	Implementation of Secure Storage Backend Component	Implement the prototype. Adapt the REDIS platform to work with SGX hardware. This phase will compise the centralised solution and a replicated Master-Slave	3.1	Prototype
3.3	Development of testbench client for phase 1 testing	Implement a benchmark tester to interact with the platform and gather all necessary metrics such as latency, throughtput, memory consumption, cpu load and others	3.2	Prototype
3.4	Secure Transport Layer	Implement and secure the transport layer. All comunications from and to the system should be secure	3.3	Prototype Communication
4	**Testing of Prototype - Phase 1**	Using the benchmark application to gather the metrics and document them on the two different implemented architectures - Centralised and Master-Slave	3	Experimental Evaluation
5	**Development and Prototype Implementation - Phase 2**	Prototype second phase development	4	Prototype
5.1	Setup of testing environment for Phase 2 - Stack and Software	Final adaptation of the developement environment stack to work with the prototype phase two	4	Prototype Setup
5.2	Extension of phase 1 solution for a clustering replication architecture	Extend phase one prototype to work with a Clustering Replication Solution. Implement na event bus to replicate writing operations on one server to all others	5.1	Prototype
5.3	Extension of testbench client for phase 2 testing	Modify and extend, if needed, the benchmark test application to gather addictional metrics particullar to this arquitecture	5.2	Prototype
6	**Testing of Prototype - Phase 2**	Using the benchmark application to gather the metrics and document them on the new implemented architecture - Cluster	5	Experimental Evaluation
7	**Deployment of solution into a cloud environement**	Move from development environment to a production environment. Deploy the prototype to a cloud provider	6	Production Environment
8	**Testing of final prototype in production environment - cloud**	Using the benchmark application to gather the metrics and document them on the new production environment	7	Experimental Evaluation
9	**Dissertation Report Writing**	Dissertation Writing	1,2,3,4,5,6,7,8	Dissertation Writing
9.1	Dissertation Writing	Continious writing of the dissertation along the development of the prototype. This should be a continiuos effort to make sure all aspects and details are captured and	-	Dissertation Writing
9.2	Partial Report revisions (per chapter)	Revise the written work chapter by chatper at the end of some key points along the development phase	2,3,4,5,6,7,8	Dissertation Writing
9.3	Final Report revision	Final dissertation report revision across all chatpers	8	Dissertation Writing
10	**Publication (GitHub) of developed prototypes and demos**	Make the prototype available to the public by a publication on a open source platform - GitHub	6	Prototype Publication
11	**Final Exam: Dissertation Presentation**	Prepare the dissertation and the power point presentation for the final evaluations	9	Dissertation Presentation
11.1	Preparation of presentation	Produce a power point presentation of the work performed during the dissertation for a visual help on the final presentation	9	Dissertation Presentation
11.2	Finalization of presentation for the final exam (jury)	Final revision of the whole thesis, along with the power point presentation and preparation for the final exam	11.1	Dissertation Presentation

Figure II.1: Work Plan Details