HW-SW architectures for security and data protection at the edge

Tesi di Laurea Magistrale in
Ingegneria Informatica

Author: **Donald Gazidedja**

Student ID: 10591781
Advisor: Prof. Stefano Zanero
Co-advisors: Christian Pilato
Academic Year: 2021

Contents

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor Assoc. Prof. Stefano Zanero, Asst.Prof. Christian Pilato, Dott.Ing. Senni Valerio and Dott.Ing. Fabio Federici who guided and supported for the whole period of the thesis. Without them, it wouldn't have been possible! A million times thank you!

My sincere appreciation goes to the whole Polimi Como Campus team (professors, student secretary, etc.), it was an amazing experience for as long as it lasted and I am so very thankful for your time!

I want to thank my second family, my friends, for the beautiful moments we passed during these years and for the support we gave to each other to reach our goals, especially my brother from another mother, Amrit.

Last but not least, I dedicate this thesis to the most important persons, my family members and my fiancee for their support during my study period in Italy, even tho far away, they have always been there for me in every moment!

Abstract in lingua italiana

Con il rapido sviluppo della tecnologia, oggigiorno, garantire la sicurezza si traduce in diverse sfide progettuali, imposte dalle caratteristiche uniche di questi sistemi. Queste funzionalità vengono eseguite su un'ampia base di elaborazione non affidabile, che include il sistema operativo, l'hypervisor, il firmware e l'hardware. La condivisione di risorse hardware e software è rischiosa e il problema della base di calcolo non attendibile consente l'accesso in diversi modi a un utente malintenzionato per compromettere l'applicazione / hypervisor / sistema operativo per rubare segreti dal sistema.

Ani fa, ricercatori e aziende utilizzavano tecniche basate su software per fornire sicurezza e isolamento in domini distinti. Ma la storia suggerisce che le vulnerabilità del kernel e del sistema operativo vengono scoperte e sfruttate più spesso. Queste vulnerabilità possono portare a un attacco di escalation dei privilegi, con il rischio che il sistema perda la riservatezza e l'integrità dei dati. Per risolvere il problema della base di calcolo non attendibile, è stato introdotto il concetto di Trusted Execution Environment che si basa sull'hardware per isolare gli ambienti.

In questo lavoro analizzeremo dal punto di vista della sicurezza, quattro diverse tecnologie di ambienti di esecuzione attendibili che sono TPM, ARM TrustZone, AMD SEV (inclusa un'introduzione a SEV-SNP) e Intel SGX. Inoltre, dimostreremo e spiegheremo diverse funzionalità e componenti dell'architettura per capire come possono proteggere da varie vulnerabilità e quali sono le differenze o le somiglianze tra loro. Verrà eseguita un'analisi comparativa basata su una serie di vulnerabilità selezionate. Le lacune e la valutazione identificate guideranno la selezione di una tecnologia specifica per sperimentare l'implementazione di una piattaforma sicura affidabile. L'attenzione dovrebbe essere posta sull'adozione e la convalida di garanzie di sicurezza ad alta sicurezza (ad esempio, utilizzando componenti con un ingombro ridotto, certificati, verificati) e su una valutazione preliminare dell'overhead prestazionale. Oltre alla sperimentazione, l'altro risultato di questo lavoro sarà la formulazione di promettenti sfide di ricerca per le specifiche tecnologie utilizzate nella campagna di sperimentazione.

Keywords: Trusted Execution Environment, Rich Execution Environment Intel SGX, ARM TrustZone, AMD SEV, Trust, Trusted Os, Secure Storage, Encryption, Enclave, Sealing, Attestation

Abstract

With the rapid development of technology, nowadays, ensuring security translates into several design challenges, imposed by the unique features of these systems. These features are being executed on a large untrusted computing base, which includes the operating system, hypervisor, firmware, and hardware. Sharing hardware and software resources is risky and the untrusted computing base problem allows access in several ways for a malicious attacker to compromise the application/hypervisor/OS to steal secrets from the system.

Originally, researchers and companies were using software-based techniques in order to provide security and isolation in distinct domains. But history suggests that kernel and OS vulnerabilities are discovered and exploited more often. These vulnerabilities can lead to a privilege escalation attack, risking the system to lose the confidentiality and integrity of the data. To resolve the untrusted computing base problem, it was introduced the concept of the Trusted Execution Environment relies on hardware to isolate the environments.

In this work we are going to analyze from a security perspective, four different trusted execution environments technologies that are TPM, ARM TrustZone, AMD SEV (including an introduction to SEV-SNP), and Intel SGX. Furthermore, we will demonstrate and explain different functionalities and architecture components in order to understand how they can protect from various vulnerabilities and what are the differences or similarities between them. A comparative analysis based on a set of selected vulnerabilities will be performed. The identified gaps and assessment will drive the selection of a specific technology to experiment with an implementation of a trusted secure platform. The focus should be put on the adoption and validation of high-assurance security guarantees (e.g., using components with a small footprint, certified, verified) and on a preliminary evaluation of performance overhead. Besides the experimentation, the other outcome of this work will be the formulation of promising research challenges for the specific technologies used in the experimentation campaign.

Parole chiave: Trusted Execution Environment, Rich Execution Environment Intel SGX, ARM TrustZone, AMD SEV, Trust, Trusted Os, Secure Storage, Encryption, Enclave, Sealing, Attestation

1 | Introduction

A secure environment must guarantee secure operations even if there are untrusted entities. To provide this, it may be easier to design a secure embedded system if we can rely on the existing physical security of the device or come up with an assumption that, secure parts of the system cannot be accessed by malicious entities. However, embedded systems are sometimes required to work under complex trust relationships, where one party wants to put sensitive data and functions on the hands of another, with the assurance that the second party cannot modify them, if not required by the entity who owns the data. Security measures can be implemented at hardware-level or at software-level, with the objective of preserving the system's key security properties (along the three axes: confidentiality, integrity, and availability) despite upper-level vulnerabilities.

Bypassing the security measures can have enormous impacts, one of the main reasons why this can happen is because of the weak security architecture that leads to possible vulnerabilities. The OS Kernel is a part of the Trusted Computing Base (TCB) of many systems, this means that a vulnerability in the kernel, will lead to unsecure systems, and it will open the door to an adversary to bypass the protection mechanisms. Later he can compromise the system, gain root access, and get data that are supposed to stay secure. Researchers have been trying to find different ways and methods to support the isolation and manage the resources of applications, analogous to that provided by the OS. Butler Lampson, 1974 *mentions that harm can be inflicted in several ways, from destroying or modifying other users' data, reading/copying data without permissions till a denial-of-service attack. [30]* A way of mitigating the risk is having different protection domains/environments where each domain has its own capabilities, and can/cannot access certain data. As mentioned by Gang Tan, *since the introduction of protection rings and virtual memory in MULTICS, all modern operating systems are structured to have an OS protection domain, also known as the kernel mode, and multiple user-application domains, which are processed in the user mode. The OS domain executes privileged instructions, sets up virtual memory protections, and performs access control, this is the "protection" from the user-application domain, which must go through the OS domain via system call interface to perform privileged operations. [45]*

Another approach was presented by Stuart E. Madnick and John J. Donavan, in 1973 by combining virtual machine monitor/operating system to create system isolation that can provide better software security compared to a conventional multiprogramming operating system approach.[32]

According to the literature, a secure environment should allow the execution of arbitrary code within a confident environment that provides tamper-resistant execution to its applications. M. Sabt, M. Achemlal mention in their research that many names and similar ideas with the same goal exists, such as closed-box VM, operator Virtual Machine (OVM), TrustZone software, and trusted language runtime,[40] but in the last decade, the term Trusted Execution Environment, coined by Global Platform, is being used. The idea of Trusted Execution Environments (TEEs) was proposed, to solve issues like isolation of user applications and data from system software by providing cryptographic layers and secure isolation on the HW component to mitigate against attacks. The general idea behind every solution using TEE is to provide protection creating an isolated environment that will create secure storage, protection, and confidentiality of applications code, remote attestation to protect from observation and tampering by unauthorized parties (provide trustworthiness), and protecting against general software attacks.

The architecture is composed of 2 different domains, the Rich Execution Environment, and the Trusted Execution Environment. The Trusted Execution Environment is used to enhance the security and protect the security-sensitive data, secrets, and applications running in the TEE against malicious software in the untrusted Rich Execution Environment (REE). The isolation is enforced via embedded hardware technologies which can be built inside the processor or as "external devices" like TPM. We will be focusing on some of the most used and recent technologies such as TPM, ARM Trust Zone [36], AMD SEV (SEV-SNP), and Intel SGX[21] .

There are plenty of use cases where embedded systems are being used in daily life, for example, industries as Aerospace, telecommunication, cars are using it to provide some gadgets or equipment's, where we can mention some of the most common ones, Smart Watches, IoT devices, Smart Homes, Digital phones, Automatic toll system or even Industrial Robots, etc. [17]

1.1. Background and Motivations

The goal of this work is to provide a better security approach to understand the recent TEE technology introduced by Intel for the new versions of its processors called Secure Guard Extension (SGX). This work is based on some existing resources that describe this technology, including the Intel white paper and reports, workshop papers from Intel researchers and research works done by community.

This work tries to tailor various aspects ranging from a TEE high-level architecture to understand the concept, a sort of comparative analysis between the technologies and their adequacy w.r.t. various critical vulnerabilities, to lower levels, like understanding how different components communicate inside and outside Intel SGX. The practical work starts with the definition of a use case, using Intel SGX to provide hardware security, and trying to understand the outcomes and limitations of this technology. Our use case is based on a Cloud scenario and is created by different actors. Using Intel SGX, we should be able to provide trust from the data source (IoT devices), protect the data, secrets, and source code from the attackers. Intel SGX uses Enclaves, which is the environment where all the security applications will be running. We explore Sealing and Attestation to improve the security of our data and try to understand better how Intel SGX can help to mitigate and protect data from various vulnerabilities that target entities on normal execution environments or embedded systems.

The practical work requires a machine with Intel SGX available in the CPU and BIOS. For this thesis, we decided to explore Intel SGX on Ubuntu 20.04. At first, it was quite challenging to put all the puzzles together, because of lack of documentation, and not all the available sample codes or tutorials are available or updated for our OS. We even run into some bugs when trying to run sample codes provided from SGX SDK, which required help from Intel SGX forum. This also helped the community because it was a bug that came with this version of SDK for Ubuntu 20.04.

1.2. Problem Overview

On our devices, normal applications and specific applications which need to run in a secure environment, are being executed on a large computing base. Being composed of many important entities, this large computing base includes the Operating System, hypervisor, firmware, and hardware, and this makes it more complex and harder to provide security. Usually, Operating Systems have thousand and millions of lines of code, and this means more vulnerability to be exploited. According to CVE [3] , from 2015 till 2021, Linux has around 574 vulnerabilities, out of this around 205 are from Code Execution, 73 on Memory Corruption, 200 on Information Gain, and 111 on Gain Privilege. And from 2018 there is a decrease in these vulnerabilities.

From 2010 and March 2011[20] , 141 Linux kernel vulnerabilities were discovered, and the studies suggested that a kernel bug should be treated as security-critical and patched [15] as soon as possible. The outcome stays the same, the attacker can gain root privilege, mount attacks by corrupting important kernel data or by hijacking control flow to existing kernel code etc.

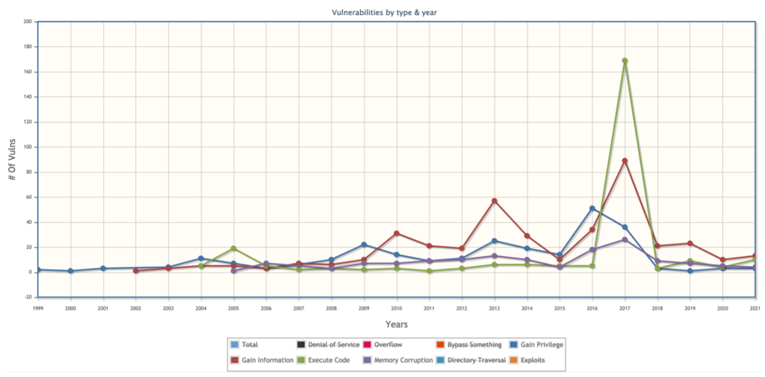


Figure 1.1: Linux Vulnerability by type and year

The main problem we are addressing is how to provide security on our data and code? Usually trusted and untrusted applications are being executed in a shared environment, using shared resources, and being controlled by untrusted entities. This opens the way for malicious to attack and exploit the vulnerabilities and gain unprivileged access, maybe corrupt the hypervisor or the OS. The kernel permissively exports a sensitive interface to all users, allowing unprivileged users to alter crucial system state and information. For example, CVE-2010-1146 and CVE-2010-4347 are two vulnerabilities on Linux where the attacker successfully gains root privilege, by modifying the kernel's control flow editing directories which are supposed to be private. LXFI [33] is a system proposed in 2011 to partition the modules through module principal idea in Linux. And the goal is to prevent an adversary from exploiting vulnerabilities in kernel modules in a way that leads to a privilege escalation attack.

What the researchers have been trying to do in the past 50 years, is to create an isolation layer using different domains to prevent privilege escalation, data leakage, protection of memory regions, code corruption, etc. Securing the environment with hardware-based security will provide more protection since the secure execution environment is isolated from the rest of the system and the attack surface is smaller.

According to Gruschka and Jensen, the attack surface is categorized between the user, service, and cloud providers.[25] Figure 1.2demonstrates the attack surface when the system does not use Inter SGX. While Figure 1.3 demonstrates how Intel SGX minimizes the attack surface to protect the data in a secure environment. By creating 2 domains, code and data on the secure domain cannot be accessed from outside and the memory regions are separated. Intel SGX provides strong data integrity and authorization by using sealing and attestation.

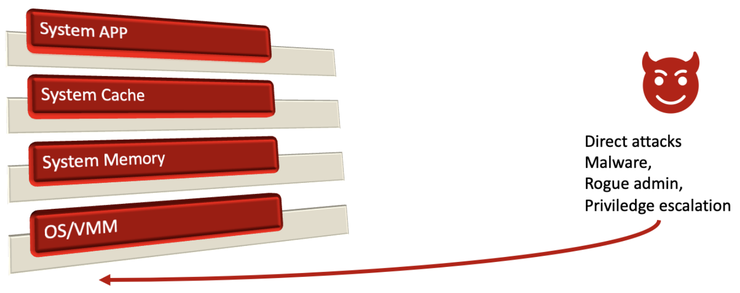


Figure 1.2: Attack Surface Without Intel SGX

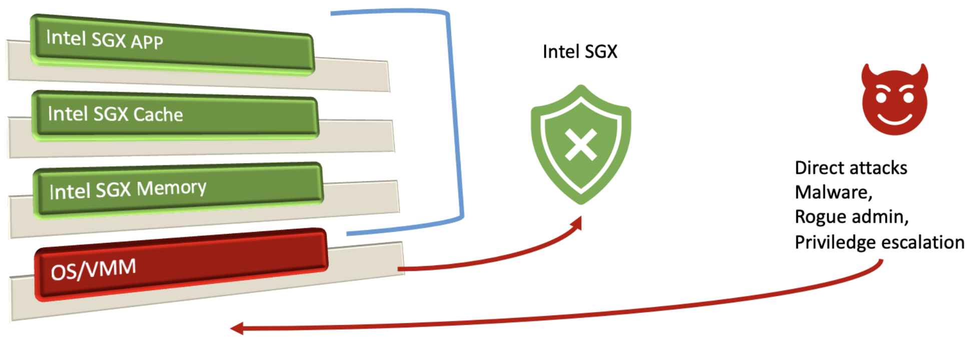


Figure 1.3: Attack surface using Intel SGX

1.3. Results

First, a selected list of vulnerabilities from CWE is analyzed w.r.t TEE technologies, focusing mainly on important aspects such as confidentiality and integrity of the data. The goal was to understand how this vulnerabilities impact each of the HW/SW components and if protection is provided or these technologies are vulnerable analyzing their common and different features regarding the HW and SW component. The conclusion was that Intel SGX and TPM provide better protection than ARM TrustZone and AMD SEV, because of their architecture and using features such as attestation and sealing.

Second, we analyze different HW and SW element for each architecture. The results in this section show that TPM uses two types of memory to provide security, ARM TZ and Intel SGX share the same concept using the system memory to create trusted domains. While AMD SEV uses virtual memory and physical to. Regarding encryption, Intel SGX and AMD SEV used to provide stronger algorithms, generating unique pair of keys per

VM or Enclave. But with the new update of TPM in 2019 now it uses SHA-1 and SHA-256 for hashes. Also to provide signature generation and verification plus public-key cryptography, TPM uses the RSA and ECC with Barreto-Naehrig 256-bit curve and a NIST P-256 curve. ARM TZ did not had encryption before the new version, and it uses AES-256. One very important part to provide good security is integrity. TPM and Intel SGX provide integrity in different ways, such as Intel SGX uses enclaves to provide integrity and TPM does it through the concept of Root of Trust. ARM TZ and old version of AMD SEV did not provide integrity, but the new AMD SEV-SNP does it by giving access to the data only to the "owner" who wrote them on the memory. According to my research, on overall Intel SGX and TPM provide better security because they give the opportunity to use attestation and sealing.

The expected experimental part shows on the practical point of view how Intel SGX's features work on a real life scenario. On one side, enclave provides some mechanism to provide attestation, in order to "trust" third parties. In our case it the idea to be used to attest IoT devices. Intel SGX provides two types of attestation, Intel® Enhanced Privacy ID (Intel® EPID) Attestation and Elliptic Curve Digital Signature Algorithm (ECDSA) Attestation. Differences between this two are mentioned on Subsection 2.5.1. We used EPID, because of technology limitations, ECDSA usually is available only on Intel server processors. And we show how this idea can be used on a Cloud Solution for hospitals where we have many actors and devices that need to interact, for eg: using EPID for Medical IoT device attestation, described better on chapter 4.

1.4. Thesis Structure

The remainder of this paper is organized as follows. SubSection 1.1 describes in short the goal, the motivation of this work, and the background on Intel SGX. SubSection 1.2introduces with the problem that we want to overcome by giving a simple example on Linux OS vulnerabilities, Subsection 4.2 describes the attacker model to identify the threats and specify the security requirements, Subsection 3.2.2 we make a classification of some vulnerabilities selected from CWE, targeting embedded systems, then we give a brief explanation how each technology protects and what can be the outcome of the attack. Chapter 2 introduces the TEE and explains the idea behind it using a reference architecture and its components. Here we understand better the notion of Trust between different entities. In this chapter we also describe the architecture of TPM, ARM Trust Zone, AMD SEV, and we focus more on Intel SGX where we explore some of its main components and functionalities. Chapter 3 describes a mapping table with different components from each technology and gives a description on how each technology provides with the following elements. We have selected some critical vulnerabilities on the TEE technologies and the focus is to understand which of the technologies is affected and if not, which component ensures the mitigation and how. Chapter 4 explores the use case, by describing a high-level architecture, the different actors and how we will use the Intel SGX. When designing an application using Intel SGX it is recommended to follow the following approach: 1) Identify the application's secrets. 2) Identify the providers and consumers of those secrets. 3) Determine the enclave boundary. 4) Tailor the application components for the enclave. On Chapter 5 we the conclusion and possible improvements on this work.

2 | Review of literatures

Summary

We start this work by describing first the TEE concept and what is the main goal of this technology. A reference architecture is provided to create the main idea of trusted and untrusted executed environment which is composed of different elements and functions that are briefly described in subsection 2.1.

This chapter will describe and compare TPM, ARM Trust Zone, AMD SEV, and Intel SGX[21]. Four different TEE architectures with the same mission: provide a secure domain. In the beginning, we will describe the main designs, components, and futures and in the end, we will talk about vulnerabilities and different mitigation methods. Special focus will be given on Intel SGX which is quite a new technology with only 5 years but has taken special attention from a lot of researchers.

Disclaimer: We will not go into much detail in the TPM, ARM Trust Zone, and AMD SEV, but we will give a high-level description of the main functionalities. What we will focus more on is Intel SGX. Intel SGX has been gaining attention from researchers not only because it is quite a new technology, but compared to the mentioned solutions, it provides a unique idea to use what so-called Enclave in the trusted execution environment and it uses attestation and sealing. An Enclave is the smallest entity, which is responsible for running the trusted applications and saving the data to the secure memory area. Many use cases have been proposed using Intel SGX and we focus our research on providing a solution on a complex system such as the cloud, including different actors and external IoT devices to provide with user's data. Intel SGX can be also used combined with other software solutions to secure secrets or code from the untrusted environment.

2.1. Trusted Execution Environment (TEE)

In order to create strong security mechanisms in embedded systems, some companies create their own architecture design by using different types of components, logic, and domains. Even if we assume that each part of the system is well secured, the way that they are composed can lead to exposure to new vulnerabilities. The responsibility to create a secure environment is big because the end product will be sold to companies that provide services based on it,

Trusted Execution Environment is an architecture used to create trusted establishment solutions that provide secure and isolated environments from potential attacks. It provides a level of assurance for the following properties:

1. Data Confidentiality: Unauthorized entities cannot view data while in use within the TEE.
2. Data Integrity: Unauthorized entities cannot add, remove, or alter data while it is in use within the TEE.
3. Code integrity: Unauthorized entities cannot add, remove, or alter code executing in the TEE.

This is ensured by isolating memory areas to execute code and store sensitive data that can be accessed only by trusted elements using hardware mechanisms that those other environments cannot control. It provides security features such as isolated execution, integrity, and confidentiality of the trusted applications. Both domains utilize several different components such as RAM, ROM, Memory, Cryptographic accelerators, etc. It has secure storage where confidentiality, integrity, and freshness of stored data are guaranteed and where only authorized entities can access the data. And in case there is a need for data to travel from one domain to another the communication between the two is done through Access Control which provides components hardware software to check the communication between the partitions.

TEE works on the notion of trust, inside the security domain there are trusted entities, the component which is responsible for this is Root of Trust. Root of Trusts starts at the beginning of technology, which comes from manufacturing with trusted elements, and during the runtime, it is its duty to define elements on trusted or not trusted. There are different techniques, but the most two important are Dynamic and Static Trust (Mohamed Sabt, 2015) [40].

There is always a drawback, TEE cannot be fully protected against hardware-based vulnerabilities and software ones. Because of the different components and entities that create the whole architecture, there will be weak links in the system which an attacker can exploit and perform successful attacks. Even in modern times, where the technologies are more complex and secure, some attacks have been successfully performed and this leads to big concerns, for example, TPM-Fail using timing and lattice attacks to recover 256-bit private keys, Foreshadow[46] attack on Intel SGX targeting enclaves operating within an untrusted context, Spectre [26] and Meltdown[31], etc. Section **??** describes a reference architecture and goes more into detail about how TEE works and what are some of its main elements.

2.1.1. Reference Architecture

Different companies configure TEE architecture, hardware, and software, based on their requirements in order to achieve maximum security. Such components or logic may be for example use of volatile/nonvolatile memory, separation of system memory in the secure and non-secure domains or using separate memory, use of Root of Trust, use of Certified Authority, use of RAM, ROM, the configuration of secure boot, etc. More will be described for each technology. As mentioned before, the goal of TEE is to provide a secure environment so sensitive data/code/functionalities can run there. This way the system has an isolation layer and access to secure memory is restricted only to trusted entities. The communication between the two environments is performed through TEE Access Control which is responsible to check and trust the API calls from the Rich Execution environment (untrusted part) toward the TEE and vice versa. Now we go more in detail on how this works, and why this architecture provides good protection.

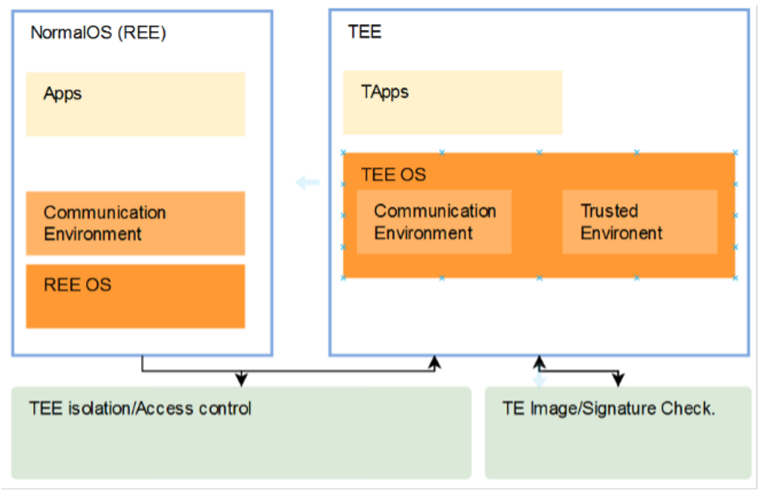

NormalOS (REE)
Apps
Communication Environment
REE OS
TEE
TApps
TEE OS
Communication Environment
Trusted Environent
TEE isolation/Access control
TE Image/Signature Check.

Figure 2.1: High-End TEE architecture

Normal Operating System is the untrusted execution environment.

REE Rich Operating System Execution Environment – Originally considered to hold the Rich OS(s). Now, due to the deployment of TEE technology in IoT environments where the "other OS" may not be considered feature-rich, the REE may also be described as holding the Device OS(s). In one sense, it should be considered "everything else" inside the device that is not part of the current TEE and hence capable of attacking the TEE. In terms of security, the REE should include components such as Secure Elements and other TEEs if present, but for simplicity, these are kept separate.

Rich OS The "main" operating system inside a device. Such an OS is designed to provide the user access to the general software functionality of a device and is considered "feature-rich". In a TEE-enabled device, this Rich OS runs in the REE, alongside the TEE and may make use of some services offered by TA's.

Apps The applications that run on the Rich OS. These apps should not be able to access the data, update, delete or manipulate on TEE.

REE Communication Provides support for messaging between the Client Application and the Trusted Application.

Trusted Execution Environment.

Is an environment that is isolated from where our trusted code/applications are executed, and the data is stored. It has a signature-check all loadable components and a run-time environment where only signature-check application software is loaded. For different technologies, the TEE is assigned in different ways. A TEE is composed of software and hardware components.

Trusted OS The Trusted OS is the component that exposes run-time capabilities to enable and manage Trusted Applications. In IoT devices, this may not be an OS as such, but a set of generic services, such as a TLS stack enabling secure communications to off-device entities. **TEE OS** has all the management components such as, Secure Storage, Trusted I/O paths, secure attestation, etc.

Secure Storage Is where all the data are saved and cannot be accessed from untrusted entities but only from authorized ones. Data should be saved with minimum encryption security and integrity or confidentiality should be provided. Sealed storage is an example that has integrity-protected secret keys, accessible only from TEE.

Cryptographic and data rollback protection mechanism.

Secure Boot ROM is used to start the secure boot, and it is accessible only from TEE. At the moment of boot, TEE is isolated from the REE. The secure boot is carried out in five steps (Arfaoui, Gharout, and Traoré, 2014) [14].

- The boot starts from the trusted ROM.
- The first initializations are done, and the TEE Trusted OS is authenticated and validated.
- The TEE Trusted OS is set up. Then it prepares the environment to boot the REE.
- The REE takes control and starts the REE OS initializations.
- The REE is set up and the TEE functionalities are ready. Thus, the mobile is turned on.

TEE Communication Environment Agent: a driver in TEE that enables safe communication between REE and TEE. But this introduces new threats such as: message overload attacks, user control (Duongsaa, Jun. 2005.),[39] and data corruption attacks. Three models of communication are mentioned in literature (1) Global Platform TEE Client API, (2) secure RPC (Remote Procedure Call) of Trusted Language Runtime, and (3) real-time RPC of SafeG.

TE Image/Signature Check

ontains components that check and assign the validity of TEE components. It can also contain elements such as Root of Trust (RoT) which is a combination of software, hardware, and data, providing a service for which no other combination of software, hardware and data is capable of attesting the initial state of the system.

TEE Isolation/Access Control provides components hardware software to check the communication between the partitions. The communication succeeds only if it passes the security checks. It also provides Secure Scheduling which guarantees a “balanced” and “efficient” coordination between the TEE and the rest of the system. This way we are sure that the tasks running in both Environments do not affect each other and TEE does not affect the responsiveness of the main OS.

2.2. Trusted platform modules (TPM)

TPM [16] has been around since early 2003, created by a group of computers engineers who came to be known as Trusted Computer Group (TCG). The first TPM 1.1b[41] was created to be physically attached to the motherboard of the PC, and since it had to be cheap and cost-effective everything that could be done by the software was not included in the HW. The goal of TPM 1.1b was to provide key generation (limited to RSA keys), storage, secure authorization, and device-health attestation. Since it was a new technology in the market, not every pc vendor was ready to adopt it easily. It lacked compatibility at a hardware level, it required different drivers, pins, etc. But it also suffered from quite a few attacks e.g., dictionary attacks, to guess the password. The TPM version 1.2 was released with some change's w.r.t to 1.1b, and their major target was to protect from dictionary attacks. It also included a non-volatile RAM of 2kb so the keys would not get lost when switching devices. Years later TCG introduced TPM 2.0, which we will describe more in, as an updated version with some important changes in the encryption algorithms. TPM 1.2 used the SHA-1 algorithm but it was weak, and it needed a change to protect against brute force and cryptoanalysis.

As described in the main white paper , TPM is a device that enables trust in computing platforms in general. The main goal is of this technology is to provide a secure environment, inaccessible from the external environment, for sensitive data and to establish the foundation of trust between different entities. TPM architecture can vary depending on the platform that it will be installed, so the architecture is Slightly different. These applications make the authorization of secrets more secure and allow only entities with proper authorization to access them. In this work, we will explore the general TPM architecture.

TPM was first designed as a separate hardware component, and it was more like a root of trust for the system where it was embedded but can also be designed as part of the CPU. It used to communicate with the CPU through the bus with Lpins (check this).

Before giving a high-level description of TPM's main functionalities we will have a look on the architecture and describe the main components.

2.2.1. TPM Architecture

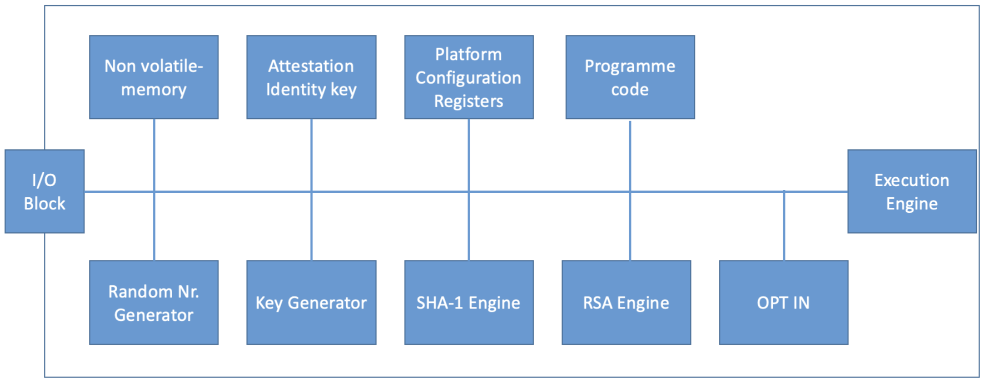


Figure 2.2: TPM Architecture Example

In order to provide security TPM works on the notion of trust, it should trust the entities on which require or want to perform some functionalities inside the TPM. So TCG has defined schemes for establishing trust on the platforms composed of HW and SW components so they can provide identification. TPM relies on a concept called Root of Trust, and the different trusted platforms combined with Root of Trust perform the security check. According to TCG cannot be proved that Root of Trust is behaving properly, but they can check how different roots have been implemented by checking the certificates. A platform should have 3 Root of trust,

Root of trust for measurement which send integrity-relevant information to the RTS and is controlled by the Core Root of Trust for Measurement (CRTM) which is the first set of instructions executed when a new chain of trust is established which is the starting point for root of trust. At this moment its values are sent to the RTS.

Root of trust for storage makes sure that the shielded memory cannot be accessed by any entity other than the TPM. And in this case, TPM acts as the Root of Trust for Storage.

Root of trust for reporting is used to report the platform characteristics. It communicates with the Root of trust for measurements and in order to provide trust to each other they by using asymmetric aliases (endorsement keys).

Root of trust is also used to attest external entities as trusted ones. They are different types of attestation used, but the general concept is to provide trust for an external entity to communicate with TPM. TPM provides protection on data and operations that should run in a safe environment, it uses two types of memory RAM and Non-Volatile memory. Ram is used to hold transient data in TPM, which can be deleted, and Non-Volatile memory holds the secrets. Both RAM and NVM data cannot be accessed from unauthorized entities.

2.2.2. Vulnerabilities

In the beginning, we mentioned that each pc provider can configure the TPM architecture based on its needs, it chooses to use or not different functionalities. So basically, do it at your own risk, this means more vulnerabilities, or more security. But this wasn't the case for Intel fTPM and STMicroelectronics TPM chips. These Vulnerabilities are also mentioned on CVE, CVE-2019-11090 impacts Intel's Platform Trust Technology (PTT). CVE-2019-16863 impacts the ST33 TPM chip made by STMicroelectronics. The reason behind this vulnerability is "timing leakage." An external observer can record the time differences when the TPM is performing repetitive operations and infer the data being processed inside the secure chip – all based on the amount of time the TPM takes to do the same thing repeatedly.

A passive adversary who observes signals can reconstruct cryptographic keys and break the confidentiality and authenticity of a computing system. Because of physical phenomena such as power consumption, electromagnetic emanations, or timing behavior.

- Side-channel attacks are a potential attack vector for secure technologies like TPMs. These attacks exploit the unregulated physical behavior of a computing device to leak secrets.
- Low key encryption

2.3. ARM TrustZone

Arm TrustZone [11], is an efficient way to provide security using hardware-enforced isolation built inside the CPU. Compared to TPM, ARM comes inbuild into the CPU. Trust Zone is based on TEE architecture where the environment is divided into 2 different execution environments, the Normal World, and the Secure World with system-wide isolation between them. As it is shown in Figure 3.2, the architecture of ARM TrustZone is similar to the classic TEE, also Intel SGX uses the same approach. Before going into more details, Arm TrusZone has many different architectures, what we will be focusing on are Armv8-M and Armv8-A. First, we will give a simple understanding of Arm TrustZone then make a high-level comparison between the two.

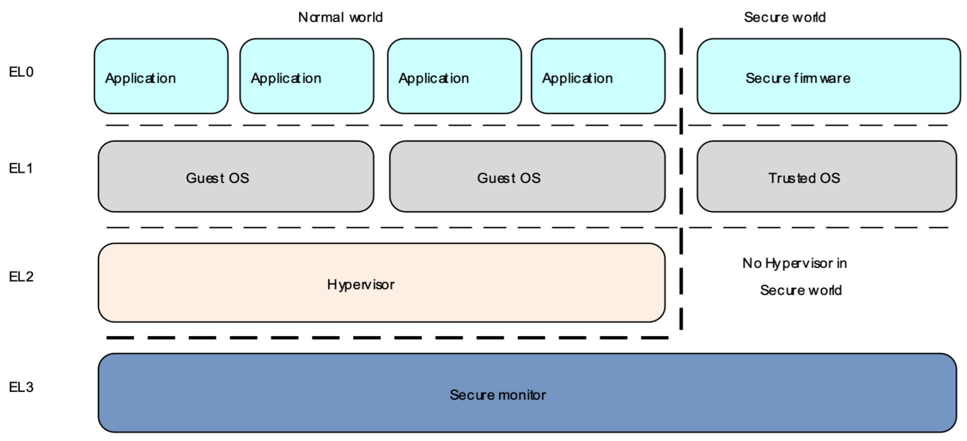


Figure 2.3: Arm TrustZone Architecture. From: Security in ARMv8-A systems, v1.0, 2021, p.5 [2]

On a high-level description, the normal world is used to run complex system software, like Linux operating system, hypervisor, etc. As mentioned in TEE, this environment can be an easy target for aggressors, since it is a complex software stack the size of the attack surface is larger. Introducing the Secure world, the architecture will have a smaller attack surface, and the system secrets can be saved into secure memory. The trusted secure world runs smaller and simpler software applications, which are also trusted by the environment. The creation of a separate secure world means that we also need to have other secure entities, one of them is the memory. The trusted memory is isolated from the untrusted environment and cannot be accessed, modified and it is divided by means of an additional bit, called NS bit. Something which should be mentioned is that in ARM TrustZone the level hierarchy is the opposite to Intel SGX, it starts from L0 as lowest to L3 as highest.

2.4. AMD SEV

So far, the technologies we described, including intel SGX next, create a secure execution environment by separating a physical part of the environment. AMD SEV does it by means of virtualization, it isolates the VMs from the hypervisor and it encrypts the memory of VMs by taking advantage of SME (Secure memory encryption) technology. AMD SEV evolved during the years, and now there is SEV-SNP (secure nested pages). AMD SEV suffered from integrity attacks; it couldn't provide protection against hypervisors that would attempt to steal the data from the protected memory regions!

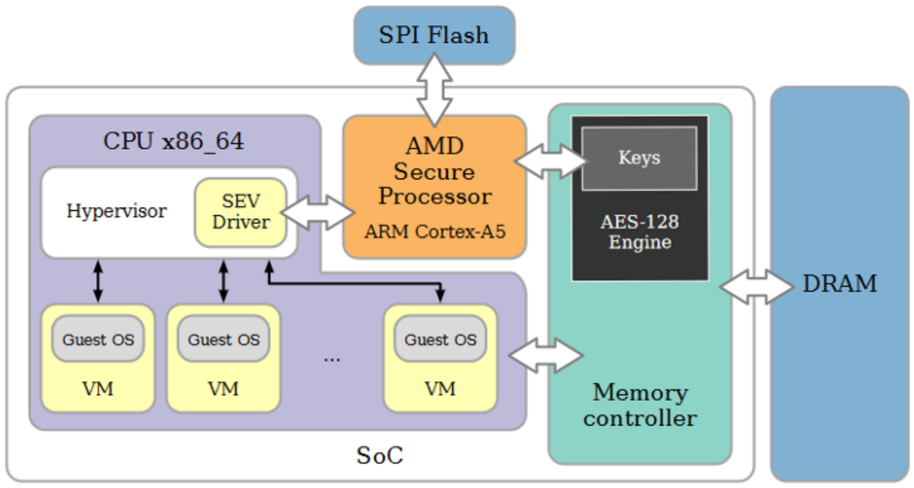


Figure 2.4: AMD SEV Architecture, From: R. P. Pires. Distributed systems and trusted execution environments: Trade-offsand challenges.CoRR, abs/2001.09670, 2020, p.9. [38]

SME is a real-time memory encryption technology, and it makes the memory more resilient from some types of attacks such as snooping or cold boot attacks. By using SME, SEV provides memory encryption for each VM, by providing unique keys generated in the Secure Processor isolated from the rest of the system-on-chip. This means that theoretically, only this VM can access its data, even the hypervisor.

When it comes to sharing data and communicating with hypervisor and OS, AMD SEV uses a C-bit, in each page table of each page entry to indicate if a page has been encrypted or not. This way the VM and Hypervisor can use the unencrypted pages to share data.

Integrity

As mentioned in the description of AMD SEV, this version lacks integrity attacks, where a determined attacker can change the value of data in the memory even though he cannot have the encryption key. The VM will now see random data in the memory. This type of attack can be mitigated somehow by the software inside the VM, but the Software inside VM doesn't know if its data has been compromised and this makes it hard to be tracked.

AMD SEV-SNP is the upgraded version of AMD SEV which includes integrity in the VM software. The idea behind SNP is that the VM should read the lattes value it wrote in the memory area and nothing else, otherwise it will give an exception and the value will not be read. This means it provides integrity protections against these threats Replay Protection, Data Corruption, Memory Aliasing, and Memory Re-Mapping.

SEV-SNP does this by enforcing entities only to write their own memory pages, and it is done by using the Reverse Map Table mechanism. About memory Aliasing and Memory Re-mapping, we will see how AMD SEV-SNP protects in the next section 3 where we map different vulnerabilities with the selected TEE technologies.

2.4.1. Privilege Levels

Privilege levels in AMD are introduced with the new version of AMD SEV-SNP and are of the nature where VMPL0 (Virtual machine privilege level) is the highest privilege level and VMPL3 is the least privilege. Each VM has a vCPU which is assigned a VMPL, and the RMP entry for each page of private guest memory is also augmented with page privilege so we have standard paging permission.

2.5. Intel SGX

Intel SGX was launched in 2015 from Intel, to provide a secure hardware technology inside the CPU, unlike TPM which is placed in the motherboard usually near the CPU. It used the same basic idea of TEE to create two separate environments where the secure environment is used data integrity, authenticity, confidentiality, etc. Intel SGX uses some different and similar concepts compared to the above-mentioned technologies and from some other similar technologies in the market. Two main concepts to protect data inside the Secure Environment are attestation and sealing which are used to provide security between two entities or reuse the data after reboot. This is done using Enclaves, an execution environment responsible for running code inside the secure environment, isolated from potentially malicious OS or hypervisor. Even though SGX provides a secure layer from the untrusted part of the system, like from hypervisor or the OS, it was proven and admitted by Intel that it is not resilient from side channels attacks.

Figure 2.5 is a reference architecture for Intel SGX, demonstrating more in detail how it works, different functionalities, enclave lifetime, memory security, Attestation, encryption Libraries, etc. Chapter 4 describes the work done based on Intel SGX.

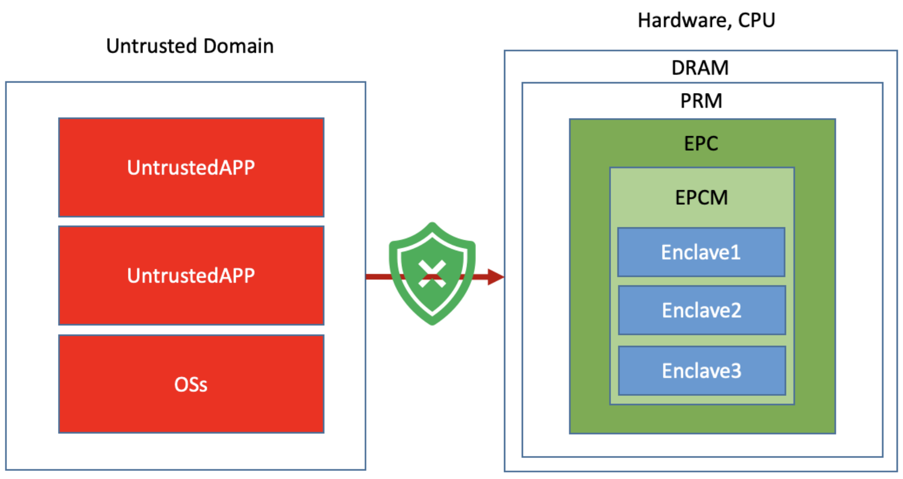


Figure 2.5: Intel SGX High-level architecture design

2.5.1. Intel SGX Architecture

Memory Organization

Processor Reserved Memory (PRM) is the part of the memory where code and data are saved from Intel SGX, it is a subset of DRAM and it belongs to the secure execution environment. The data stored in PRM cannot be accessed by entities on the untrusted environment even the CPU itself including system software, and System Management Module Ring 2. The picture below is the composition of Secure memory.

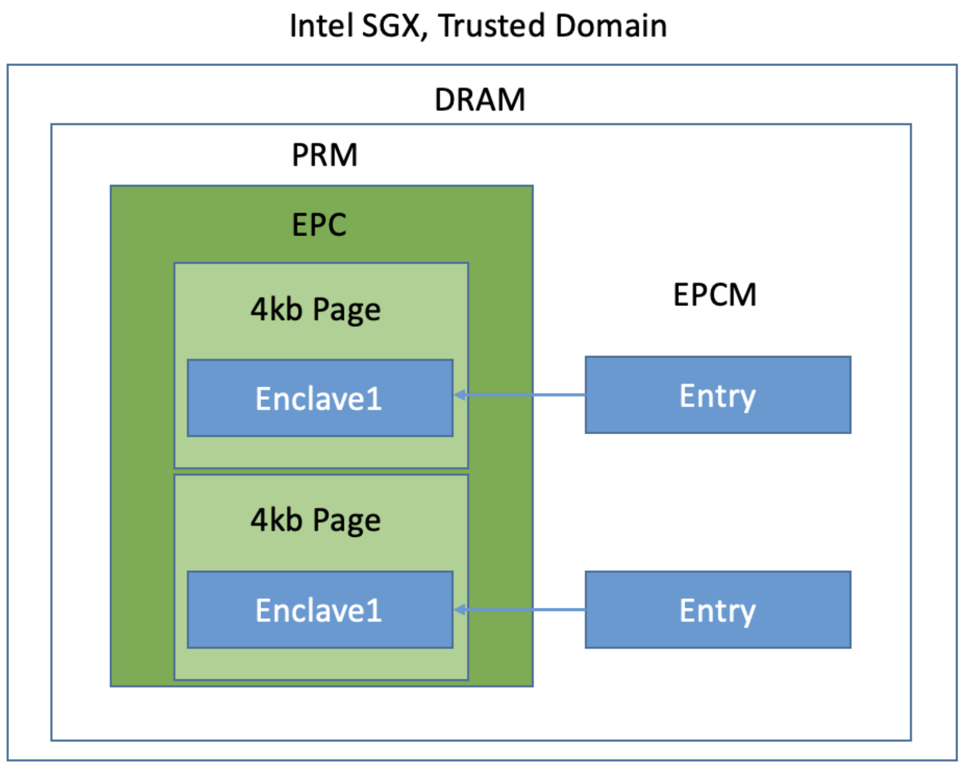


Figure 2.6: Intel SGX Memory Organization

Enclave Page Cashe (EPC)

As a subset of PRM we have EPC, which is used to store the data structures and content of enclaves. Each EPC is composed of pages of 4kb and each of those is designed to be assigned to enclaves, so we can have multiple enclaves running in the system. Because the total size of PRM is 128 MB max, the number of enclaves we can have is limited. The management of EPC is done by the same system software that manages the rest of the computer's physical memory which can be an OS kernel or a hypervisor, but it uses the SGX instructions to allocate unused pages to enclaves.

Since the system software is not trusted by Intel SGX, the SGX processor will check if the mapping of an EPC page with an enclave is done correctly, otherwise, it will refuse to perform any action that will compromise the security of the trusted system. To do these security checks, it uses the Enclave Page Cashe map, which saves the allocation decision of the system software. EPCM has 3 fields on which it saves the information and tracks the ownership, as shown in table 2.1. The PT flag determines the use of the EPC page if it will be used for the enclave or other entities such as the Enclave Control Structure.

Enclave's EPCM		
Field	Bit	Description
Valid	1 BIT	0 for un-allocated EPC Pages
PT	8 BIT	Page type.
ENCLAVESECS		Identifies the enclave owning the page.

Table 2.1: Enclave Page Cashe map, which saves the allocation decision of the system software. Each enclave is mapped to one EPC, and this table describes the fields that EPCM needs to provide the security checks.

From a security point of view, SGX should not allow operation on EPC pages with a valid bit of 1 in the EPCM, to protect from different vulnerabilities on memory region overlapping or from editing another enclave's EPC Page. The ENCLAVESECS, saves the enclave that owns the page, this information is used to enforce SGX's isolation guarantees and to prevent enclaves from accessing each other's private information. If two enclaves will communicate, it is done through eCalls and oCalls, and not via EPC pages.

Enclave Attributes

As mentioned before the EPC page can be used from different elements in the SGX and the PT flag should be set correctly. SGX stores per-enclave metadata in the SGX Enclave Control Structure, in a dedicated EPC page with a page type PT_SECS.

Enclave's identity is almost synonymous with its SECS (SGX Enclave Control Structure). The first step in bringing an enclave to life allocates an EPC page to serve as the enclave's SECS, and the last step in destroying an enclave deallocates the page holding its SECS. The EPCM entry field identifying the enclave that owns an EPC page points to the enclave's SECS. The system software uses the virtual address of an enclave's SECS to identify the enclave when invoking SGX instructions.

All SGX instructions take virtual addresses as their inputs. Given that SGX instructions use SECS addresses to identify enclaves, the system software must create entries in its page tables pointing to the SECS of the enclaves it manages. However, the system software cannot access any SECS page, as these pages are stored in the PRM. SECS pages are not intended to be mapped inside their enclaves' virtual address spaces, and SGX-enabled processors explicitly prevent enclave code from accessing SECS pages.

To run an enclave, it is important to set the correct values on the enclave's SECS. In case a programmer, on the moment that he starts the enclave execution environment, sets the wrong values, he may risk opening an attack gate. For example, if the DEBUG field is set to true when an enclave is running in production, it enables the DEBUG features of SGX such as the ability to read and write most of the enclave's memory. According to Intel SGX Developer document, the recommendation is to set all the attribute flags, except Mode 64-bit, Provision Key, and Launch the key, and none of the XFRM attributes.

For example, the Mode 64 bit should not be touched, because it will make the enclave run into a 32-bit environment and will change its behavior which should be avoided, for security reasons.

Enclave's SECS		
Name	Size	Description
DEBUG	1 BIT	Debugging features, r/w enclave's mem. But in the development environment.
XFRM	64 BIT	Running with XCR0 register. Specifies the set of architectural extensions to produce enclave code.
MODE64BIT	1 BIT	64-bit enclave arch.

Table 2.2: Enclave execution attributes. The execution environment depends on the configuration of these attributes.

Enclave Life Cycle

When creating an enclave, SGX manages and prepares the whole system, allocates EPC to the enclave, etc. The life cycle of an Enclave goes into different stages, where first is the creation of an enclave, loading, initialization, and teardown.

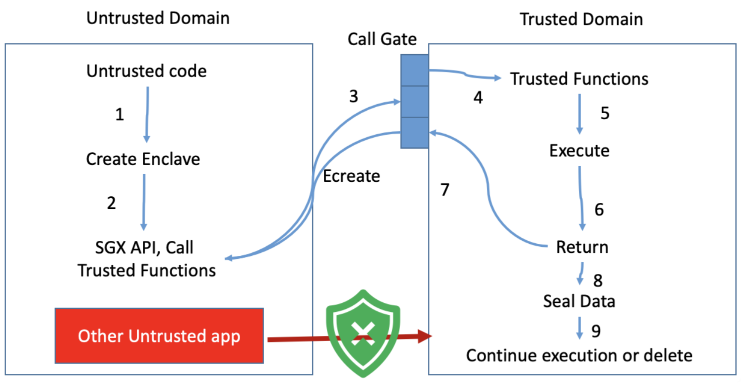


Figure 2.7: Enclave Life Cycle

Ecreate: To create an enclave, the function is called from the untrusted part of the app. An important part of the system is the EDGER8 tool, which provides edge routines to define the interface of the untrusted app and the enclave. By reading the EDL file of the enclave, this function will create two parts, Enclave_u.h, enclave_u.c which contains prototype declaration and function definition for the untrusted proxies and bridges. Enclave_t.c contains function definitions for trusted proxies and bridges, Enclave_u.h prototype declarations for trusted proxies and bridges.

EADD/EEXTEND: EADD instructions are used to load the initial code and data into the enclave. EADD is used to create both TCS pages and regular pages. This function copies a source page from non-enclave memory into the EPC, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in EPCM. EADD reads its input data from a Page Information (PAGEINFO) structure.

Initialization (EINIT): This is the last function for the enclave creating process. After EINIT, the MRENCLAVE measurement is complete, and the enclave is ready to start user code execution using EENTER instruction. When EINIT completes successfully, it sets the enclave's INIT attribute to true. This opens the way for ring 3 application software to execute the enclave's code, using the SGX instructions. On the other hand, once INIT is set to true, EADD cannot be invoked on that enclave anymore, so the system software must load all the pages that make up the enclave's initial state before executing the EINIT instruction.

TearDown

After the enclave has done the computation, it was designed to perform, the system software executes the EREMOVE instruction to deallocate the EPC pages used by the enclave. EREMOVE will first make sure that there is no logical processor executing code inside the enclave and then will mark the EPC page as available, so it will be freed for use.

Attestation and Sealing

Sealing The last function in the enclave lifecycle will destroy it and its EPC. This can lead to data loss and no other enclave can read others enclave stored data. In order to provide a more stable solution, sealing is used. This allows secrets to be retrieved if the enclave is torn down (either due to a power event or by the application itself), and subsequently brought back up. There are two types of sealing Sealing, the first one seals the data to the Enclave Identity and the second one will seal the data to the Sealing Identity. The first one will allow data to be accessed by enclaves sealed by the same sealing authority and the second can sign multiple enclaves with the same key and allow them to exchange data.

Attestation Attestation is the process of demonstrating that a piece of software has been properly instantiated on the platform. Intel® SGX is the mechanism by which another party can gain confidence that the correct software is securely running within an enclave on an enabled platform. So basically, when two enclaves want to communicate, they can trust each other by providing secrets generated

Local Attestation and Remote Attestation. The successful result of local attestation provides an authenticated assertion between two enclaves running on the same platform that they can trust each other and exchange information safely, while remote attestation provides this kind of verification for the ISV client to the server so that ISV server can confidently provide the client with the secrets it requested. Intel SGX provides two different types of Attestation key algorithm, Elliptic Curve Digital Signature Algorithm (ECDSA) and Intel Enhanced Privacy ID (Intel® EPID).

Data center Attestation Primitives using Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm [42] This attestation service is provided by Intel to support non-Intel attestation infrastructures for Intel SGX. This provides with an enclave called Provisioning certificate Enclave which acts as a local CA for local Quoting Enclaves which are running on the same platform. Quoting Enclaves provides the PCE with the public key and it issues a certificate identifying the QE and the Attestation Key. This structure is signed by the Provisioning Certification Key. Intel Provides APIs so the communication between different entities is done in a secure manner. Figure 2.8 shows how the Data Center Attestation Primitive works to generate and attest a third party.

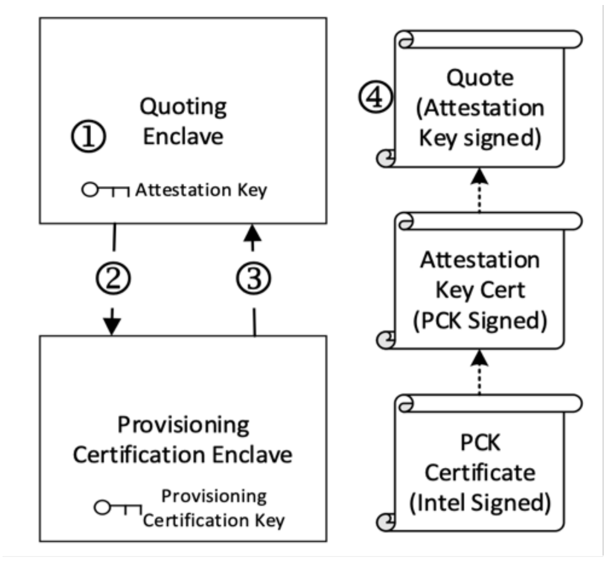

Quoting Enclave
①
Attestation Key
②
③
Provisioning Certification Enclave
Provisioning Certification Key
④
Quote (Attestation Key signed)
Attestation Key Cert (PCK Signed)
PCK Certificate (Intel Signed)

Figure 2.8: Attestation cycle for third parties. From: V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski. Supporting third party at-testation for intel sgx with intel data center attestation primitives.White paper, 2018, p.3 [42]

The ECDSA attestation key generated by the QE needs to be certified by an Intel® SGX key rooted to the platform HW fuses. Intel develops and signs an enclave called the Provisioning Certification Enclave (PCE). The key generated by the PCE to certify (sign) attestation keys is rooted to the CPU HW fuses. This key is called the Provisioning Certification Key (PCK) private key. Intel will also generate and publish a public key that matches the signing key (PCK) generated by the PCE. The public key is published as an X.509 certificate format called the Provision Certification Key Certificate (PCK Cert). The PCE will provide an interface to retrieve the PCK Certificate identifier (EncPPID+TCB+PCEID) used by a verifier to find the matching PCK Cert. The PCE also provides a mechanism to sign another enclave (i.e. QE) REPORT using the PCK private key. For Intel® SGX DCAP, the QE will generate the ECDSA Attestation Key (AK) and include a hash of the AK in the QE.REPORT.ReportData. Only the PCE can produce the PCK private key. This PCE certification data will ultimately be embedded in the ECDSA Quote generated by the QE. The AK is then used to sign application enclave Reports to prove that the enclave is running with Intel® SGX protections at a given TCB. This is called the ECDSA Quote. The Attestation infrastructure owner can verify the ECDSA attestation key using the PCK Certificate.

Attestation using Intel Enhanced Privacy ID (Intel® EPID) [18] EPID is a group signature scheme that allows a platform to sign objects without uniquely identifying the platform or linking different signatures where each signer belongs to a “group”, and verifiers use the group's public key to verify signatures. EPID supports two modes of signatures, fully anonymous mode of EPID a verifier cannot associate a given signature with a particular member of the group, while in n Pseudonymous mode an EPID verifier has the ability to determine whether it has verified the platform previously. As in DCAP, quoting enclaves create EPID key used for signing platform attestations which are then certified by an EPID backend infrastructure. The key is the reason Intel SGX can trust the platform of the underlying hardware. Figure 2.9 shows an example of how an application with a secure processing element on the user platform could provide an attestation to a challenging service provider in order to receive some value-added service from the provider. Note that many usages will use this process infrequently (e.g., at enrollment time) to provision the enclave with a communication key that will then be used directly in subsequent connections.

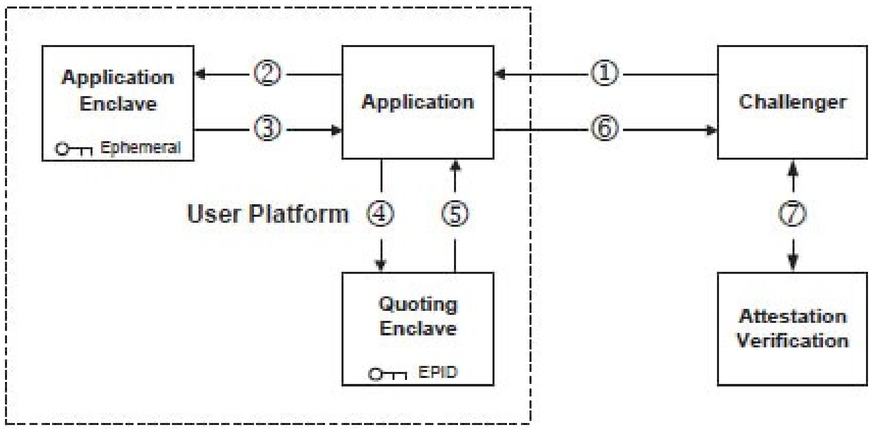


Figure 2.9: Remote Attestation Example From: V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski. Supporting third party at-testation for intel sgx with intel data center attestation primitives.White paper, 2018, p.3 [42]

1. Initially, the application needs service from outside the platform and establishes communication with the service-providing system. The service provider issues a challenge to the application to demonstrate that it is indeed running the necessary components inside one or more enclaves. The challenge itself contains a nonce for liveness purposes.

2. The application is provided with the Quoting Enclave's Enclave Identity and passes it along with the provider's challenge to the application's enclave.

3. The enclave generates a manifest that includes a response to the challenge and an ephemerally generated public key to be used by the challenger for communicating

secrets back to the enclave. It then generates a hash digest of the manifest and includes it as User Data for the EREPORT instruction that will generate a REPORT that binds the manifest to the enclave, as described in section 3.2. The enclave then sends the REPORT to the application.

4. The application forwards the REPORT to the Quoting Enclave for signing.
5. The Quoting Enclave retrieves its Report Key using the EGETKEY instruction and verifies the REPORT. The Quoting enclave creates the QUOTE structure and signs it with its EPID key. The Quoting Enclave returns the QUOTE structure to the application.
6. The application sends the QUOTE structure and any associated manifest of supporting data to the service challenger.
7. The challenger uses an EPID public key certificate and revocation information or an attestation verification service to validate the signature over the Quote. It then verifies the integrity of the manifest using USERDATA and checks the manifest for the response to the challenge it sent in step 1.

Crypto Library

Crypto API Toolkit for Intel(R) SGX (CTK) aims at enhancing the security of data and key protection applications by exposing interfaces that run the key generation and cryptographic operations securely inside an Intel(R) Software Guard Extensions (SGX) enclave. We will use this library to encrypt our data with high-level encryption API.

SGX SSL [5]

It provides cryptographic services for Intel SGX. This will be useful when we pass the data from the trusted execution environment to the untrusted one. Using its api, one can use Ocall in a more secure way. Intel® SGX SSL library provides integrity and confidentiality of security assets and protects them from both malicious software and a simple hardware attack.

Figure 3.10 shows how Intel SGX SSL uses a trusted library providing an implementation for missing system APIs inside an enclave and an untrusted library providing implementation of missing system APIs outside an enclave. The untrusted part calls the trusted code with a function in the EDL file of enclave and based on the API which is called the trusted code continues execution.

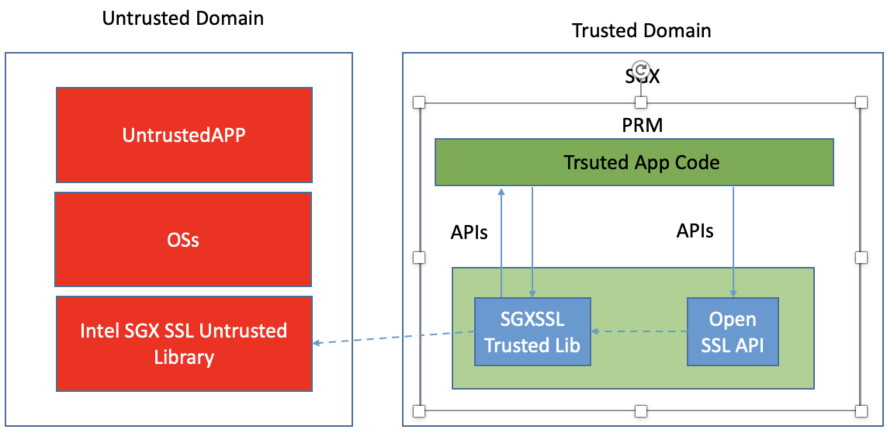


Figure 2.10: SGX SSL, Providing a secure communication between two domains

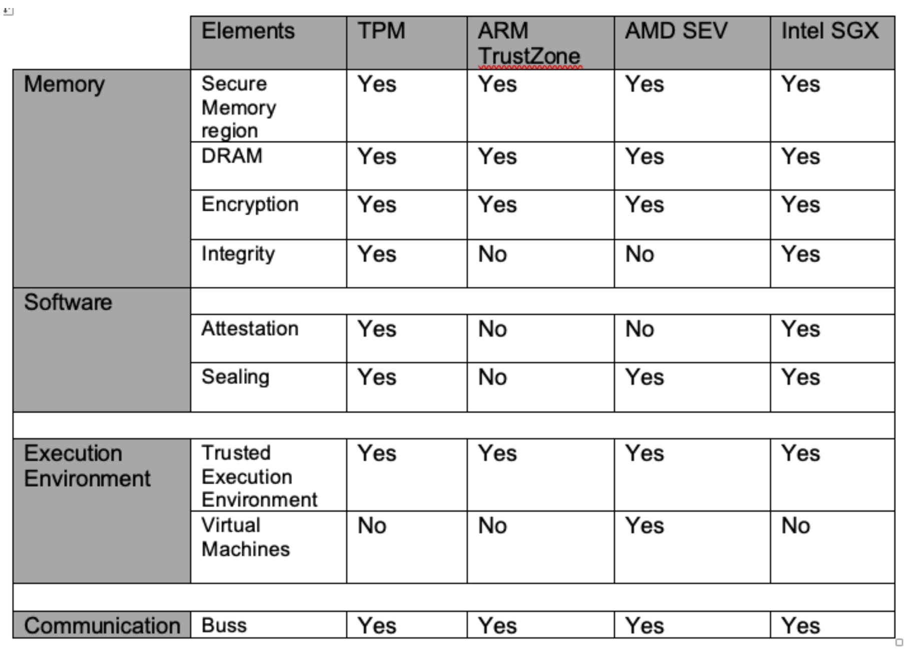
3 | Technology Research & Assessment

Summary

Different TEE technologies are built on similar but quite different HW/SW architecture, each trying to provide the best security solutions. The goal of this chapter is to provide a unique mapping table of HW/SW elements for each technology, a description on what is the role, etc.

3.1. Architecture Elements

The focus is given on four main elements: memory, software, communication, and execution environment. We select some entities and critical functions, and this will be correlated with Section 5 where a more detailed description of some selected vulnerabilities is given on how they impact the system and the protections against them.



	Elements	TPM	ARM TrustZone	AMD SEV	Intel SGX
Memory	Secure Memory region	Yes	Yes	Yes	Yes
	DRAM	Yes	Yes	Yes	Yes
	Encryption	Yes	Yes	Yes	Yes
	Integrity	Yes	No	No	Yes
Software					
	Attestation	Yes	No	No	Yes
	Sealing	Yes	No	Yes	Yes
Execution Environment	Trusted Execution Environment	Yes	Yes	Yes	Yes
	Virtual Machines	No	No	Yes	No
Communication	Buss	Yes	Yes	Yes	Yes

Figure 3.1: Comparison of physical elements and functions that compose the technologies

We believe that Intel SGX provides better security compared to the other technologies, not only because it creates a good isolation environment, but it also provides confidentiality and integrity of Code and Data inside the enclave. While most of these technologies aim to create isolation only for the secrets. Intel SGX code integrity can be leveraged by other software companies, which want to protect and run important code inside the enclave. Recent research used Intel SGX to isolate tensor flow code and data which may contain private and sensitive information [28]. Something else that makes Intel SGX more "secure", is the fact that it uses attestation and sealing as describer on section 3.4.2. In a case where we must trust elements from outside the trusted environment or even inside, one can use attestation to provide trust in certain elements. For example, in a scenario where we must pass data between two enclaves, they should be attested in order to trust each other. Furthermore, the memory region inside the trusted execution environment is well protected and cannot be accessed from untrusted entities. This protects the data from unwanted privilege escalation attacks [23].

Memory

- **Secure Memory** is a safe location to store the data in the trusted execution environment.
  - TPM has two types of Secure memory Volatile (store temporary data and code related to the executing application) and Non-Volatile (store persistent data

related to integrity of the system and its users and associated state).

- ARM TZ has a physical memory that is shared between trusted zones and the normal world, the partition can be done physically or virtual at booting time by a secure firmware. This is done by the Memory Protection Unit MPU which is a programmable unit that allows privileged software, typically the OS kernel, to define memory access permission. Regions of the physical memory are designated as secure which can be accessed only by TZ.
- AMD SEV has a virtual memory and physical which is secured by SEV. Virtual memory is used to access data within the virtual memory space while Physical addresses are used to directly access main memory.
- IntelSgx Uses DRAM to create a subset of it for the secure memory region. This subset is called Processor Reserved Memory (PRM) and it is used to store Enclave Page Cache (EPC), where all enclaves are created inside this region. Usually, the PRM is 128MB (EPC takes 93.5MB) but it can be configured between 32MB, 64MB.

- **Encryption**
  - TPM technology uses a random number generator system for encryption. But this system is not reliable and the newer version uses a more sophisticated algorithm.
  - ARM TZ does not provide cryptography techniques alone, and so it needs some software or hardware crypto accelerators such as Arm TrustZone CryptoCell products [1]. One security solution that can be used is Arm CryptoCell Family, respectively CryptoCell-300 and 312 for lower power and low area design and CryptoCell 700 and 712 .
  - AMD SEV uses Secure encrypted virtualization and Secure memory encryption. SEV allows the association of one encryption key per hardware virtual machine in a way that the hypervisor has no longer access to everything within the guest VM. SME defines a simple and efficient architecture capability to encrypt the main memory and it is integrated into the CPU architecture, scalable from embedded to high-end server workloads, and requires no application software modifications.
  - Intel SGX uses the memory encryption engine (MEE). The MEE is responsible for providing cryptographic operations. At every boot, it generates random

keys every, one for cryptographic operations and another for message authentication codes (MACs).

- **Integrity**
  - TPM uses SRTM (Static Root of Trust for Measurements) and DRTM (Dynamic Root of Trust for Measurements) to provide integrity. SRTM is used on system boot aka the BIOS boot block will measure the BIOS and send the value (hash) to the TPM in a location called Platform Configurations Register (PCR) 0 before executing it. Then the BIOS measure the next thing in the boot chain and again will store the value in a PCR of the TPM. This process is executed for each component in the boot sequence (PCI option ROM, boot loader, etc.). DRTM is different from SRTM, since it happens when the system is running.
    - TPM also uses the Root of Trust where the elements but be trusted and it is done with the help of Root of Trust for Measurement (RTM), Root of Trust for Storage (RTS), and Root of Trust for Reporting (RTR).
  - ARM TZ and AMD SEV does not provide memory integrity which leads to attacks to exploit this vulnerability.
  - AMD SEV-SNP . The VM should read the lattes value it wrote in the memory area and nothing else, otherwise, it will give an exception and the value will not be read. This means it provides integrity protections against these threats Replay Protection, Data Corruption, Memory Aliasing, and Memory Re-Mapping
  - IntelSgx Memory integrity and freshness in conjunction with confidentiality guarantees, on the other hand, make the secure environment robust against all kinds of memory tampering. Creating an integrity table on the memory. The main elements to provide this are MEE and MACs.

Software

- **Attestation**
  - TPM Attestation is the action of having the TPM sign some internal data with a Key. It involves the generation of a signature using the respective Attestation identification key stored on PCR (integrity matrix).
  - Arm TZ Does not provide attestation, this can lead to data leak if an attacker exploits.
  - AMD SEV one of the protection that firmware offers to protect SEV-enabled

guests, is the attestation of a launched guest, and the confidentiality of the guest's data. Attestation of the guest launch proves to guest owners that their guests securely launched with SEV enabled.

- Intel SGX stands out from the others because the code that is used only contains the private data in computation and the code that operates on the enclave. The link of trust starts from a signing key owned by the HW manufacturer (acts as a CA) which shall be trusted by the verifier. The manufactures provisions each secure processor that produces a unique attestation key, which is used to produce attestation signatures. The manufacturer also issues an endorsement certificate for each secure processor's attestation key.

- **Sealing**
  - TPM sealing the data is encrypted and it is associated with a PCR state, and it will only be decrypted if the PCR value is the same as at the time of the encryption.
  - Arm TZ and AMD SEV do not provide sealing.
  - Intel SGX sealing depends on the platform hardware key derivation and is obtainable through the EGETKEY instruction.

Execution Environment

- TPM commands (instructions) are executed by the execution engine in a secure and reliable manner. The execution engine is an on-chip (within the boundary of a TPM) processor that provides execution isolation.
- Arm TZ has trusted zones and the normal world which is selected by the system on boot time. The same logic applies to Intel SGX, but the trust zone is predefined.
- AMD SEV provides a trusted execution environment by VMs. Which are created and encrypted based on the needs.
- Intel SGX: Executes the code inside Enclaves which are secured inside the trusted execution environment. Each enclave is mapped to an EPC and its data are safely encrypted and stored inside the memory region.

Communication

More information on the bus vulnerabilities is provided in the CWE - 1264 section.

- **TPM**, This depends on where the TPM location is [47]. TPM usually uses the LPC bus to communicate through the host and its environment, since some of them are implemented as single-chip components. The host can change the values inside the TMP only through the I/O buffer that is part of the system.

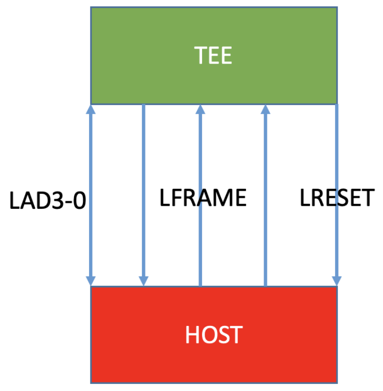


Figure 3.2: Communication of TPM using LPC bus

- Arm TZ systems connect the processors, memories, and peripherals using common bus types include AMBA High-performance Bus (AHB), Advanced Peripheral Bus (APB), and Advanced eXtensible Interface (AXI) [**?**]. In other words, the core's security state information propagates via hardware logic present in the TrustZone enabled AMBA AHB5 / APB4 bus fabric (an extra signal (HNONSEC[1] = 0) on the AHB bus indicates a secure transaction and vice versa). [7] This allows extending security to memories and peripherals through bus filters also known as TrustZone-aware peripherals which are directly connected to AHB — MPCs, PPCs, AHB/APB bridge (used as a secure gate to block or propagate secure/non-secure transaction towards APB agents). Ensuring that no secure world resources can be accessed by the non-secure world components, enabling a strong security perimeter to be built between the 2.

- AMD SEV uses an external and internal bus to provide communication with different parts of the system. The external bus for example is used to provide cache-coherency with the main memory. AXI bus is the main bus of the system to provide communication between different parts and APB is a simpler, lower power bus than the main AXI bus. The APB protocol does not carry the bits related to the TrustZone security state of the bus transactions. This places responsibility for managing the security state onto the AXI-to-APB Bridge that provides the interface between the high-speed AXI domain and the low-power APB domain. Each AXI-to-APB bridge provides an AXI slave interface and can mediate accesses for up to 16 peripherals on its local APB bus. The bridge contains address decode logic that generates the APB peripheral select based on the incoming AXI transaction. The bridge includes a single TZPCDECPROT input signal for each peripheral that is located on the bus. This signal is used to determine if the peripheral is configured as Secure or Non-secure; the bridge will reject Non-secure transactions to Secure peripheral address ranges.

- Intel SGX system bus is used for communication with the memory and the I/O devices and it connects all these components. When an outside system asks for data, Intel SGX does not provide them unless they are attested by the system, which means it should be trusted. The data are sent in a secure manner with eCalls and oCalls, using sgxssl library.

3.2. Vulnerability Model

3.2.1. Introduction

Trusted Execution Environments come with a bunch of vulnerabilities and threats which are discovered by chance or by “bad” intention from the attacker. Below is shown a list with some of them based on recent research [37]. The focus is to discover HW/SW vulnerabilities and threats which already exist and give a result on which technology can protect them better against them and why. Some similar research has been done providing some good information to understand which vulnerabilities still affect trusted execution environment technologies [35].

3.2.2. Selected classes of embedded systems vulnerabilities

In this section, we try to classify some vulnerabilities found in embedded systems based on CWE. We consider these vulnerabilities very critical for our system and if not taken into consideration a malicious user can exploit them and make significant damage to our system. The focus will be given only on HW/SW vulnerabilities, which are very common and dangerous leading to data stealing, manipulation, and breaking the integrity or confidentiality of the system.

In Chapter 3 a table of selected vulnerabilities is shown, described on the impact they have and if different TEE technologies provide mitigation or not. In this section, we go more into detail, as we explore that some components may be important not just for one or two types of vulnerability, but sometimes for a good part of them.

Some other vulnerabilities that can be exploited but are out of the scope for this work, are related to system power, voltage, current, temperature, clocks, system state saving/restoring, and resets at the platform on the CPU. If not taken into consideration when developing the architecture, they may lead to serious security issues in the system. These kinds of vulnerabilities are out of the scope for the moment since some of them require special devices to be performed.

Hardware Vulnerabilities		
Authority and Core Issues (LA) Vulnerabilities		
CWE-1302	Security Identifier	Chip/OS/VMs/TrustZone
CWE-1281	Miss-configuration of ISA and CPU Logic	CPU Logic and ISA
CWE-1252	Miss-Configuration of CPU for W/R	Memory
CWE-1220	Insufficient Granularity of Access Control	Memory/Registers
CWE-1260	Overlap Between Protected Memory Ranges	Memory Regions
CWE-1261	Register Interface Allows Software Access to Sensitive Data	Register Interface -> hardware functionality
Security Flow Vulnerabilities		
CWE-1190	DMA enabled too early in boot phase	DMA/Secure World
CWE-1264	Insecure De-Synchronization between Control and Data Channels	Buss
CWE-1274	Volatile memory containing boot code	NVM
Logic Design Vulnerabilities		
CWE-1254	Incorrect Comparison Logic Granularity	MACs
Memory Vulnerabilities		
CWE-1257	Improper Access Control Applied to Mirrored/ Aliased Memory Regions	Memory/Cache
CWE-1282	Assumer-Immutable Data is Stored In Writable Memory	Memory/Cache
Cryptographic Vulnerabilities		
CWE-1279	Cryptographic Operations are run before supporting units are ready	Cryptographic Units
CWE-325	Missing Cryptography Steps	Memory/Cache/Buss

Table 3.1: A list of hardware vulnerabilities that can affect an unsecure system

Software		
Permission Vulnerabilities		
N/A	Security Identifier	Bus/Memory/Registers
N/A	Communication Channel	Memory Bus/ Bus
CWE-283	Unverified ownership	Bus/Memory/registers
CWE-1220	Insufficient Granularity of Access Control	Bus/Memory/registers
CWE-276	Incorrect Default permissions	Bus/Memory/registers
Cryptographic Vulnerabilities		
CWE-1240	Use of risky cryptographic primitives	Bus/Memory
CWE-325	Missing Cryptography Steps	Bus/Memory
Memory Vulnerabilities		
CWE-226	Sensitive Information in Resources not Removed Before Reuse	Bus/Memory

Table 3.2: A list of Software vulnerabilities that can affect an unsecure system

Hardware

The main vulnerability we wanted to point out is the Authority and Code issues. In a system, it is important that the secure part has an identification protocol controlling who can access the data, otherwise, if an attacker gets a ring 3-0 privilege, he can inject malware that can fully run on Kernel level and can successfully have full access to memory, all CPU instructions, and all hardware. A miss-configuration of ISA and CPU can lead to microarchitectural side effects. In security, Flow Issues class vulnerabilities can happen in 2 phases, one when the system is booting which it needs to run some security checks and create secrets, two when system entities communicate. Memory Issues vulnerability class, give focus on 2 vulnerabilities where the access control and immutable data are targets of different attacks. Cryptographic vulnerabilities are an issue not only in embedded systems but in all ICT, eg. blockchain [49], cloud systems, web-based systems etc. an unsecure cryptographic algorithm is considered if it provides with a key lower than 128 bits[43]. If a system uses a weak type of algorithm it must be changed, because it will be an easy target to attackers.

Software

As mentioned in the introductory section, security in traditional systems is mostly based on the software part, where antiviruses or different architecture solutions provide protection from malicious code. The main classes we are focusing on in this part are Permission Issues, Cryptographic Issues, and Memory Issues. As it is noticeable both the SW and HW have some common vulnerabilities which can be exploited using the same techniques or different ones. In the section below, we will see how using TEE some of these vulnerabilities can be mitigated, but this doesn't mean that it will provide 100% security. As we will see, new technologies face new vulnerabilities or some old ones can be used against them, which should be taken into consideration before choosing the architecture of the system. This should protect the cloud service providers and their clients from:

- Unauthorized access of data.
  - Provide Confidentiality
- Integrity of data
  - Protecting from improper modification
- Secure Data encryption and Decryption using Trusted Execution Environment.
- Secure Code Execution

3.2.3. Mapping of Vulnerabilities

The two tables below show a map of different vulnerabilities taken from CWE. Each one has a status corresponding to each architecture.

- NV: Not Vulnerable, the corresponding technology provides minimum protection against the vulnerability type.
- Vulnerable: Technology does not provide minimum protection.
- N/A: it is not known if the technology is vulnerable or not

CWE ID	**HW Vulnerability**	**TPM**	**ARM TZ**	**AMD SEV**	**Inter SGX**	Target
CWE-1302	Security Identifier	NV	NV	NV	NV	Chip/OS/VMs / TrustZone
CWE-1281	Miss-configuration of ISA and CPU Logic	NV	NV	NV	NV	CPU Logic and ISA
CWE-1252	Miss- Configuration of CPU for W/R	NV	NV	NV	NV	Memory
CWE-1220	Insufficient Granularity Access Control	NV	NV	NV	NV	Memory / Registers
CWE-1260	Overlap Between Protected Memory Ranges	NV	NV	NV	NV	Memory Regions
CWE-1261	Register Interface Allows Software Access to Sensitive Data	NV	NV	NV	NV	Register Interface -> hardware functionality
Logic Design Vulnerabilities						
CWE-1254	Incorrect comparison Logic Granularity	NV	NV	NV	NV	MACs
Security Flow Vulnerabilities						

CWE-1190	DMA enabled too early in boot phase	N /A	NV	NV	NV	DMA /Secure Domain
CWE-1264	Miss-configuration of ISA and CPU Logic & beta	NV	NV	NV	NV	BUSS
CWE-1274	Volatile memory containing boot code	NV	N /A	N /A	N /A	NVM
Memory Vulnerabilities						
CWE-1257	Improper Access Control Applied to Mirrored/ Aliased Memory Regions	NV	V	V	NV	Memory /Cache
CWE-1282	Assumer- Immutable Data is Stored In Writable Memory	NV	V	V	NV	Memory /Cashe
Cryptographic Vulnerabilities						
CWE-1279	Cryptographic Operations are run before supporting units are ready	NV	V	NV	NV	Cryptographic Units
CWE-325	Missing Cryptography Steps	NV	V	NV	NV	Memory 1/ Cashe /Buss

Table 3.3: A list of hardware vulnerabilities that can affect an unsecure system.

CWE ID	**SW Vulnera-bilty**	**TPM**	**ARM TZ**	**AMD SEV**	**Intel SGX**	Target
			Permission Vulnerabilities			
N /A	Security Identifier	NV	NV	NV	NV	Bus/Memory/ Register
N /A	Communication Channel	NV	NV	NV	NV	Memory Bus/ System Bus/
CWE-283	Unverified Ownership	NV	NV	NV	NV	System Bus /Memory / Registers
CWE-1220	Insufficient Granularity Access Control	NV	NV	NV	NV	System Bus /Memory / Registers
CWE-276	Incorrect Default Permissions	NV	NV	NV	NV	System Bus /Memory / Registers
			Memory Vulnerabilities			
CWE-226	Sensitive Information in Resources not Removed Before Reuse	NV	V	V	NV	Memory /Cashe /Ciphertext
N /A	Memory Integrity	NV	V	V	NV	Memory /Cashe /Ciphertext
			Cryptographic Vulnerabilities			
CWE-1240	Cryptographic Operations are run before supporting units are ready	NV	V	NV	NV	Cryptographic Units
CWE-325	Missing Cryptography Steps	NV	NV	NV	NV	Memory 1/ System Bus /Memory

Table 3.4: A list of hardware vulnerabilities that can affect an unsecure system.

On the first column is the id of each vulnerability from CWE, and in the case in the id column it is written N/A, it was proposed by us.

Even though all the architectures provide good protection for most of the vulnerabilities that can risk the domains, still it is possible to break these security mechanisms, and risk confidentiality and the integrity of data. Meltdown, Spectre, and Side Channels attacks have been successfully performed and resulting in disclosure of taking the secrets from the trusted environment.

CWE - 1220 Insufficient Granularity of Access Control

Scope	Impact	Likelihood
Confidentiality, Integrity, Availability, Access Control	Modify/Read Memory, Execute Unauthorized Code or Commands, Gain privileges or assume identity, bypass Protection mechanism.	High

Table 3.5: Vulnerability short details

The product implements access controls via a policy or other feature with the intention to disable or restrict accesses (reads and/or writes) to assets in a system from untrusted agents. However, implemented access controls lack required granularity, which renders the control policy too broad because it allows access from unauthorized agents to the security-sensitive assets. Integrated circuits and hardware engines can expose accesses to assets (device configuration, keys, etc.) to trusted firmware or a software module (commonly set by BIOS/bootloader). This access is typically access-controlled. Upon a power reset, the hardware or system usually starts with default values in registers, and the trusted firmware (Boot firmware) configures the necessary access-control protection. A common weakness that can exist in such protection schemes is that access controls or policies are not granular enough. This condition allows agents beyond trusted agents to access assets and could lead to a loss of functionality or the ability to set up the device securely. This further results in security risks from leaked, sensitive, key material to modification of device configuration.

TPM To provide granularity, TPM does it by using a combination of operations such as ownership, attestation indeed keys, and PCR.

- **Ownership** To perform all the operations, the TMP needs to be under the ownership of an entity. This means that the environment will set up all the policies and a shared secret to prove ownership per user. During this process, it will generate a Storage Root key related to a particular user. Each user is unique and cannot access other users' data unless data migration. The proof of ownership is done by using protocols with the idea of providing ownership, command, and parameter authentication.

- **AIK (attestation identity keys)** AIK key is generated from a CA at the request of the TMP owner and it is stored on non-volatile storage outside TMP. The reason why we need AIK is to generate signatures on data in order to provide data confidentiality and integrity.
- **PCR** is 160-bit data that stores a generated hash per system component, which is the result of integrity measurements. Its values are secured by the RTS, and they are stored on volatile memory. On each boot, TMP calculates new values for individual PCRs and these values are then used to provide secure boot and platform state attestation.

Upon a BIOS reset, the boot sequence starts from the Core BIOS, which its integrity measurement is stored inside PCR-0, and later it is extended to include the integrity measurements of the rest of the BIOS. Next, the motherboard configuration will be measured by Core BIOS, stored on PCR-1, and next the rest of the code, ROM firmware, and ROM firmware configuration. Then the process continues to start the OS, as usual, we need to measure the integrity of OS loader code and OS code and in the end, the respective software (application code) will be checked. After each integrity measurement, the result is stored on PCR.

This way we have a trusted and reliable view of the current state of the system. PCR-0 to PCR-7 are saved for TMP, and the rest till PCR-15 is used for operating system and installed applications. The minimum number of PCR is 16. When reporting the integrity measurement, it is more secure to generate a signature on the value to avoid replay and man in the middle. If a malicious user tries to boot a different operating system or is able to compromise the operating system (i.e. installed a rootkit backdoor to an operating system), the PCR values will be different and the data will not decrypt properly. When an entity requests the attestation, it will sign the PCR value with the sealed private key. If there is a change in any of the PCR values during the system boot, the signature will not verify. This Will indicate to the requester that the system is not in the state as before, whether it is still secure or not is a decision that the requesting entity must make.

Vulnerabilities: CVE-2018-6622: Allows local users to overwrite static PCRs of TPM and neutralize the security features. [24] It allows an adversary to reset and forge PCRs when the system wakes up.

In this paper, the authors successfully perform attacks to perform end-to-end key recovery attacks. [34] Using 3 different level of privileges. And in all 3 it was possible to perform key recovery. To perform these attacks, we need to read the processor's cycle right before the TMP device starts executing-security critical function and right after its completion.

ARM TrustZone On boot time the system decides with NS bit for the secure and not secure world. In the secure world the data cannot be accessed by the non-secure world, but TZ can access the data can access memory and I/O designated for both Secure and Normal World, whereas code running in Normal World is restricted to Normal World resources. Thus, any operations on secure data or hardware must be done by Secure World on behalf of Normal World. The transition of control from Normal World to Secure World is known as a world switch. A Horizontal Privilege Escalation vulnerability arises when Trusted App exposed APIs enable untrusted processes to access or manipulate Client app provided data. For example, key-stores contain HPE vulnerabilities if a malicious CA can obtain or use keys belonging to other CAs. A malicious CA can access any data stored in memory between a target CA's requests, provided it can time its own requests before a shared TA clears the respective data and this data is retrievable though at least one exposed APIexposed that do not perform any origin checks regarding there quests (e.g., sessions). Such attacks are most damaging when CA-provided keys are cached between CA requests.[44]

AMD SEV To provide limited access within VMs, AMD uses SEV which provides 1 key per VM. Keys are managed by a secure processor (AMD SP), coordinated by the hypervisor, which is later used to encrypt memory pages. A future to protect the guest register state from hypervisor is Encrypted State. The SEV-ES VM's CPU register state is encrypted during world switches. On the first run, the hypervisor must coordinate with AMD-SP to create an initial encrypted state image for the guest VM.

Another way is to assign on each VM a privilege level, VMPL. It is identified numerically, starting from 0 to 7. The most privileged on is VMPL0 which is only read, 1 is write, 2 execute user, 3 execute supervisor and 4-7 reserved. This is used to restrict guest memory accesses. On software, AMD provides a segment protection mechanism with the ability to restrict program access into other software routines and data. There are 4 levels (0-3), high to low. And there are 3 types of privileges, Current privilege level, description privilege level, and requestor privilege level. CPL is the privilege level at which the processor is currently executing. DPL is the privilege level that system software assigns to individual segments. It is used in privilege checks to determine whether software can access the segment. It is stored in the segment descriptor. RPL reflects the privilege level of the program that created the selector, and it is used to let called programs know the privilege level of the program that initiated the call. CPL and RPL are compared by the processor to determine the effective privilege level for data access.

Attacks that protect from: flow attacks (modifying VM state), rollback attacks (restoring VM register state), exfiltration (Reading VM state).

Intel SGX On boot time Intel SGX sets aside a secure memory region called Processor Reserved Memory, which is protected from CPU from all non-enclave memory accesses. It holds EPC which stores the enclave's code and data. EPC state is tracked by CPU in the Enclave Page Cache Metadata. The data in each enclave and the ones saved to memory cannot be accessed by any other enclave or user who doesn't have the privileges.

Integrity is provided by using cryptographic constructs. Intel SGX creates an integrity tree that holds the hashes of its children.

Access control in some cases is provided also by the Launch Enclave, which in some cases may result as an unnecessary approval step to run enclaves. The idea was to make sure that the enclave's author is a trusted entity. But the same job of the LE can become by System software that has access.

To prevent malicious enclaves from using keys such as EGETKEY, SGX includes a simple access control mechanism that can be used by system software to limit enclave access to provisioning keys.

Problem is that the enclave's initial code and data is loaded by unprotected memory and that a malicious system software might abuse the PROVISIONKEY attribute to generate a unique identifier for the hardware that runs.

CWE - 1260

Isolated memory regions and access control (read/write) policies are used by hardware to protect privileged software. Software components are often allowed to change or remap memory region definitions to enable flexible and dynamically changeable memory management by system software. If a software component running at lower privilege can program a memory address region to overlap with other memory regions used by software running at higher privilege, privilege escalation may be available to attackers. The memory protection unit (MPU) logic can incorrectly handle such an address overlap and allow the lower privilege software to read or write into the protected memory region resulting in a privilege escalation attack. Address overlap weakness can also be used to launch a denial-of-service attack on the higher privilege software memory regions. **TPM** PCR and TPM ownership change so far do not provide a risk of memory address region overlap because of the access control which controls the data written to memory. If the memory address region is subject to writing from another entity, TPM will go through a security check. If the entity has the right to write that memory region then it will go on, otherwise, it will fail. But the memory region cannot be overlapped.[13] If the PCR values will be different the data will not be decrypted on every boot so even if an aggressor can manage to overlap the memory regions, get the data, won't be able to decrypt them. Unless it gets the key, which it won't be able because it just overlapped it. **ARM TrustZone** MMU is the major component of the L1 memory system. It is capable of mapping the virtual address space that is running on the processor to the physical address space that exists outside of the processor.

A direct memory access controller is the dedicated engine for moving data around the physical memory system. It can support concurrent Secure and Non-secure channels. This means that a non-secure transaction trying to program a DMA transfer to or from Secure memory will result in the transfer failing. **AMD SEV** AMD SEV divides its address space into four levels to provide hardware isolated abstraction layers within a VM for additional security controls. The privileges start from VMPL0 high to VMPL3 which gets the lowest. VPML needs to be enabled and in this case, every CPU of a VM is assigned a VMPL. Individual guest pages is also augmented with page access rights corresponding to each VMPL and can be marked as R/W supervisor-mode executable and user-mode executable. The default permission is VPML0. Each permission can be modified via RMPADJUST instruction. The restriction is that on level cannot grant more permission than it currently has, and a higher level can modify permissions for a less privileged one. Example 1: a vCPU that is executing at VMPL0 could use RMPADJUST to restrict a page or memory to be only read-write but not executable at VMPL1. Also, RMPADJUST cannot be used to grant greater permission of what is allowed by permission mask. Example 2: If VMPL1 wants to write access to VMPL2 but when permissions were set it was not specified then the RMPADJUSR will fail.[6]

Intel SGX [21] A reserved memory area is predefined at boot time and is called PRM. PRM consists of EPC allocated to enclaves. EPCM (enclave page cache map) is used to save records about the system software's allocation decisions for each EPC page. This way, if a malicious system software wants to allocate the same EPC page to two enclaves it cannot. EPCM uses the information to track the ownership of each EPC page. But two entities can communicate via untrusted non-EPC memory. Intel SGX uses an EPC page eviction. In short, it does page swapping to utilize all the resources effectively. This method is done in save mode because enclaves do not trust the system software, so SGX offers a method that can protect against a malicious OS trying to make address translation attacks. This method uses EWB instruction which evicts an EPC page into a DRAM outside the EPC and parks the page as available. Another thing that should be kept in mind is that no TLB has address translation associated with the evicted page, in order to avoid the TLB-based address translation attack.

CWE - 1264

Many high-performance on-chip bus protocols and processor data paths employ separate channels for control and data to increase parallelism and maximize throughput. Bugs in the hardware logic that handle errors and security checks can make it possible for data to be forwarded before the completion of the security checks. If the data can propagate to a location in the hardware observable to an attacker, loss of data confidentiality can occur. 'Meltdown' is a concrete example of how de-synchronization between data and permissions checking logic can violate confidentiality requirements. Data loaded from a page marked as privileged was returned to the CPU regardless of the current privilege level for performance reasons. The assumption was that the CPU could later remove all traces of this data during the handling of the illegal memory access exception, but this assumption was proven false as traces of the secret data were not removed from the microarchitectural state.

On the reference architecture of the TEE, it is mentioned that TEE Isolation/Access Control provides components hardware software to check the communication between the partitions. The communication succeeds only if it passes the security checks. This states that by default the communication between two parties will occur and only occur if the untrusted entity passes the security checks. Which implies that the proposed technologies provide it. **TPM** On TPM the data on volatile memory cannot be accessed by everyone, weather way in the case when data need to be passed from the secure memory to unsecure entities and the only interaction is through the LPC bus.[11]

LPC bus has been shown to be vulnerable to passive eavesdropping. [29] **ARM TZ** AMBA3 AXI system bus has a control signal for each r/w on the main system bus. This bus has a protocol that defines Non-Secure bits, AWPROT[1] and ARPROT[1], used to write and read with low-high value, Secure-Non-secure. These signals are set by bus masters on every new transaction and the bus/slave decode logic must interpret them that the security mechanism has not been violated. As said high values in NS bits make them nonsecure, so the NS masters cannot access secure slaves.

In order to carry 'the security and privilege capabilities' over to other memory systems and interfaces, we use logic present in the system's bus (AMBA AHB 5/APB4) fabric i.e. the privilege attribute (HPRIV) and secure attribute (HNONSEC) are carried across the internal Advanced High-performance Bus (AHB) matrix to reach memory protection checkers (MPCs), peripheral protection checkers (PPCs), and master security wrappers (MSWs) for other bus masters.

Furthermore, to move data around the system Direct Memory Access (DMA) Controller is used to move data around the physical memory system. DMAC can support concurrent Secure and non-secure channels controlled by a APB interface.

AMD SEV VMM can restrict guest CPU access to memory. SVM provides multiple protection domains which can restrict device access to physical memory on a per-page basis. The northbridge's host bridge provides different protection domains that are associated with a device exclusion vector that specifies per page access rights of the device. Devices are identified by a device ID.

AMD does an access checking when a w/r request is received on memory staples from an external host bridge port. The id of the device doing the request is mapped to a protection domain number, which selects the DEV defining the access permission for the device. In case of error handlers, AMD uses machine-check registers to control and report hardware machine-check errors. **Intel SGX** Intel SGX contains a memory encryption engine, whose role is to encrypt and authenticate the data stored in EPC. The problem is that it doesn't encrypt the addresses on the memory bus. Which can lead to possible attacks. It is not quite mentioned what type of protocols are used in Intel SGX communication bus, but in general, it is said to not be trusted but the communication bus in the secure world is secure. Intel SGX offers a certificate-based identity system that can be used to migrate secrets between enclaves from the same authority. And this provided a strong privilege security mechanism.

CWE - 1257

Aliased or mirrored memory regions in hardware designs may have inconsistent read-/write permissions enforced by the hardware. A possible result is that an untrusted agent is blocked from accessing a memory region but is not blocked from accessing the corresponding aliased memory region. Hardware product designs often need to implement memory protection features that enable privileged software to define isolated memory regions and access control (read/write) policies. Isolated memory regions can be defined on different memory spaces in a design (e.g., system physical address, virtual address, memory-mapped IO). Each memory cell should be mapped and assigned a system address that the core software can use to read/write to that memory. It is possible to map the same memory cell to multiple system addresses such that read/write to any of the aliased system addresses would be decoded to the same memory cell. This is commonly done in hardware designs for redundancy and simplifying address decoding logic. If one of the memory regions is corrupted or faulty, then that hardware can switch to using the data in the mirrored memory region. Memory aliases can also be created in the system address map if the address decoder unit ignores higher order address bits when mapping a smaller address region into the full system address. A common security weakness that can exist in such memory mapping is that aliased memory regions could have different read/write access protections enforced by the hardware such that an untrusted agent is blocked from accessing a memory address but is not blocked from accessing the corresponding aliased memory address. Such inconsistency can then be used to bypass the access protection of the primary memory block and read or modify the protected memory. An untrusted agent could also possibly create memory aliases in the system address map for malicious purposes if it is able to change the mapping of an address region or modify memory region sizes.

TPM To the best of my research TPM doesn't suffer from memory aliasing problems because it doesn't use it. **ARM TZ [11]** For memory mapping, TZ must take care to ensure that the 33-bit address space is used in such a way that data remains coherent in all the locations that are stored. The same memory location appears as two distinct locations in the address map, one Secure one Non-secure. The protection that TZ offers is from the Secure, Non-Secure flag to the cache. ARM TZ [48] suffers from cache side-channel leaking information. [27] The attack is performed based on a cache timing side channel to extract sensitive information.

This paper proposes a new attack on memory aliasing problems with ARM TZ. Capable of extracting fine-grained information from the secure world of Trust-Zone. **AMD SEV** A new system-wide data structure called reverse Map table is used to perform additional security checks on memory access. Which is a feature on the SEV version. A reverse map table is used to ensure a one-to-one mapping between system physical addresses and guest physical addresses. RMP entries contain flags indicating if the physical page is assigned to a guest at the AMD-SNP, page size, immutable flag. The integrity of RMP is maintained by restricting software manipulations.

The security mechanism of AMD requires that the pages are validated before being accessed with the PVALIDATE. After RMPUPDATE can be used by the hypervisor to unassign, reassign or remap the page and will become invalidated.

The privileges are assigned by vCPU and they restrict guest memory accesses. The only reason when full permissions are enabled is for VMPL0 using RMPUPDATE. **Intel SGX** Uses MESIF protocol, which is implemented in the CPU and in the protocol layer of the QPI bus. The SDM and the CPUID instruction output indicate that the L3 cache, also known as the last-level cache (LLC) is inclusive, meaning that any location cached by a L1 or L2 cache must also be cached in the LLC. The QPI protocol uses home agents and cache agents to make sure the memory is coherent, and the ownership is saved.

Many cache-based timings attacks against SGX enclaves have been published, and all of them are possible because of the cache-hierarchy system and because the caching of memory leaves effects in the system state which are measurable. SGX enclaves are vulnerable to cache attacks.[37]

CWE – 1282 Assumed-Immutable Data is Stored in Writable Memory

Scope	Impact	Likelihood
Integrity	Modify/Read Memory, Gain privileges or assume identity, bypass Protection mechanism and it varies by context.	N/A But mostly unlikely

Table 3.6: Table to test captions and labels.

Immutable data, such as a first-stage bootloader, device identifiers, and "write-once" configuration settings are stored in writable memory which gives the opportunity for aggressors to reprogram or update in the field. Security services such as secure boot, authentication of code and data, and device attestation all require assets such as the first stage bootloader, public keys, golden hash digests, etc. which are implicitly trusted. Storing these assets in read-only memory (ROM), fuses or one-time programmable (OTP) memory provides strong integrity guarantees and provides a root of trust for securing the rest of the system. Security is lost if assets assumed to be immutable can be modified.

TMP Sensitive data are kept in non-volatile memory which contains shielded locations which are accessed with protected capabilities (ROM, flash memory).

ARM TZ Uses a non-volatile memory to store important information such as user keys, which is managed by a non-volatile memory manager who is also in charge of OTP storage and LCS life cycle state management. On OTP are stored important data which are meant to be written once with a 0-bit next to them.

AMD SEV Data are Stored in DRAM but encrypted from SEV (secure encrypted virtualization) which separates the data through a private bit. Vms can choose which data memory pages would like to be private. The encryption key is managed by ADM-SP, which is a separate processor present on AMD SOCs. It contains RAM, non-volatile storage which is used to save the endorsement key PEK, which is used to derive the platform Diffie-Hellman in conjunction with guest OSes every time the machine is powered on or reset.

Intel SGX Is configured in such a way that all the components that generate different types of Keys, or certificates are saved in tamper-resistant hardware or ROM memory with a security bit. For example, when we create encryption keys for Enclaves, the system can save those keys in non-volatile memory. Important attributes are also saved in this non-volatile memory so in cases of reset or boots the system will not lose the data. But if something happens to the CPU, eg. the user's system dies, we cannot decrypt the data because only 1 enclave can have access to the key.

4 | Proposed Solution Architecture

Summary

As discussed in the previous chapters, leveraging the opportunity of using an isolated hardware domain technology can enhance the security to protect from Software-based vulnerabilities in the system. Intel SGX has proven to be quite a successful solution, being used in many industry solutions because it increases the security of sensitive and critical data or code. It also protects the Trusted Execution Environment from many advanced threats that compromise BIOS, system components, or user profiles with root permission.

In this chapter, we are going to talk about how we leverage the use of Intel SGX technology on a cloud-based solution for Healthcare. First we start by giving a brief introduction on the cloud and Internet of Things impact in the healthcare section towards its evolution to Healthcare 4.0 and how can Intel SGX enhance security. than a more detailed description of the use case with a high-level representative architecture of the cloud including a description of its actors and entities and towards the end some possible scenarios associated with sequence diagrams. In the end it is described the Threat Model for the use case described and then a technical background regarding the developing process, the use of the different functionalities that Intel SGX provides, and its security impact on our solution.

4.1. Introduction

Cloud Solution, Internet of Things, Big Data are revolutionizing the Healthcare domain and its whole ecosystem bringing a new era, Healthcare 4.0. With the development of the Internet of Things healthcare, many devices which measure people's health data are being used by hospitals and doctors to monitor patients remotely and take faster precautions. However, the responsibility of software maintenance and cloud security is handed over to cloud providers. Security is not the only issue, an important factor is the performance of message delivering from devices and on January 14th, Yin Zhang propose a performance isolation algorithm that can reduce the device message delay by 87%. [50] Besides the advantages of using cloud computing, there are many security challenges as shown by recent reports. Verizon's 2020 data breach investigation report shows that 15% of all breaches in 2020 happed in healthcare organizations [9]. According to a HIPPA Journal, due to ransomware attacks, in September 2020, 9.7 million healthcare records were exposed from 95 data breaches – 348.97% more than august 2020. [8] Figure **??** shows the causes of September 2020 healthcare data breaches.

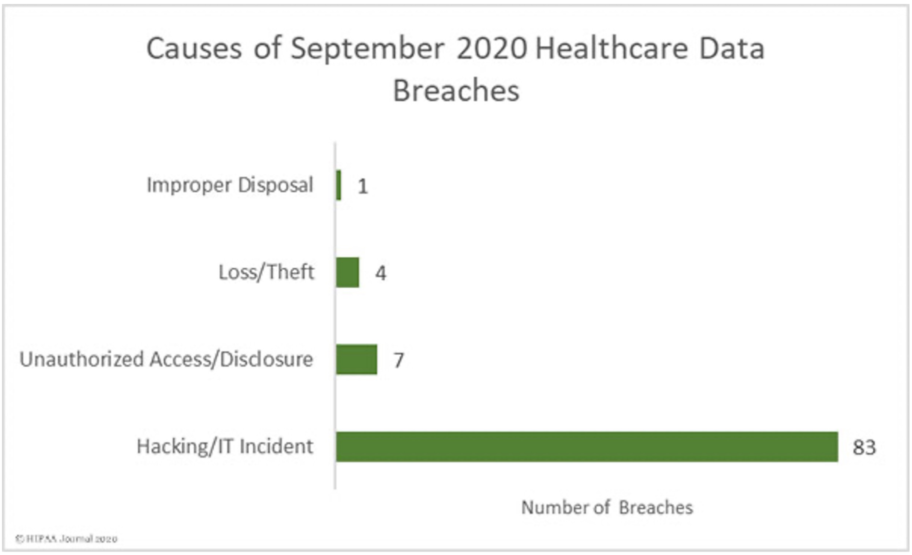


Figure 4.1: Causes of September 2020 Healthcare Data Breaches. From: HIPAA Journal on Oct 22, 2020 [8]

Since 2015 Intel has been providing a Trusted Execution Technology called Intel SGX. Intel SGX has gained quite a popularity in the past 6 years since its release, and many companies have provided useful Use Cases including here, Confidential Computing and Cloud Data Shield, BlockChain to help increase privacy and security for transaction processing, consensus, smart contracts, key storage or even Edge Computing to assist with securing IoT edge devices to cloud and client communications, etc.

Intel SGX maintains a chain of trust, for each layer that composes the system, so this way we create and provide some extra security between different domains. One of the areas that are important is the data in transit, and Intel SGX uses TLS to provide security. Section 6.4 describes the technical background of the application design based on our use case.

4.2. Attacker Model

The attacker model will describe the vulnerabilities and attacks that can happen when a system is not protected by TEE, different assumptions on the conditions an aggressor can perform an attack, and then at Chapter,3 we will see how the different technologies minimize these threats and especially Intel SGX.

In a normal situation, the security of data will be provided by using different software solutions, but this will bring the risk of security issues since more software means a larger “attack surface” and higher chances for vulnerability issues.

Assumptions: We assume that the attacker does not have physical access to the device and the only way he can access it is by remote attacks. The attacker can gain root privilege to the system and he can gain access to the ring 3 and ring 2-0 privilege levels, which he can leverage to break the confidentiality and integrity of data or inject malicious code. Ring 3 is the lowest privileged level and ring 0 is the most privileged. After gaining privilege, the attacker can perform different attacks, such as DOS attacks, data manipulation, etc. For example, a new vulnerability was discovered on Linux, CVE-2021-33909 [12], fs/seq_file.c in the Linux kernel 3.16 through 5.13.x before 5.13.4 does not properly restrict seq buffer allocations, leading to an integer overflow, an Out-of-bounds Write, and escalation to root by an unprivileged user, aka CID-8cae8cd89f05.[10] It is common that most of the architectures follow a hierarchical privilege mode. Based on this the software processes operate at different privilege levels. Because of this, security depends on OS, VMM, and BIOS levels. However, root users in the system can launch processes in kernel mode, this comes back to our problem where a malicious worker from the cloud provider can have access. This can result in data breaches. So, using the traditional security system, a malicious application running on the system may be able to use a zero-day bug or an unpatched privilege escalation vulnerability to obtain root privilege and then attack other running applications. Thus, the attack surface of a conventional application includes all the processes that are running on the same server.

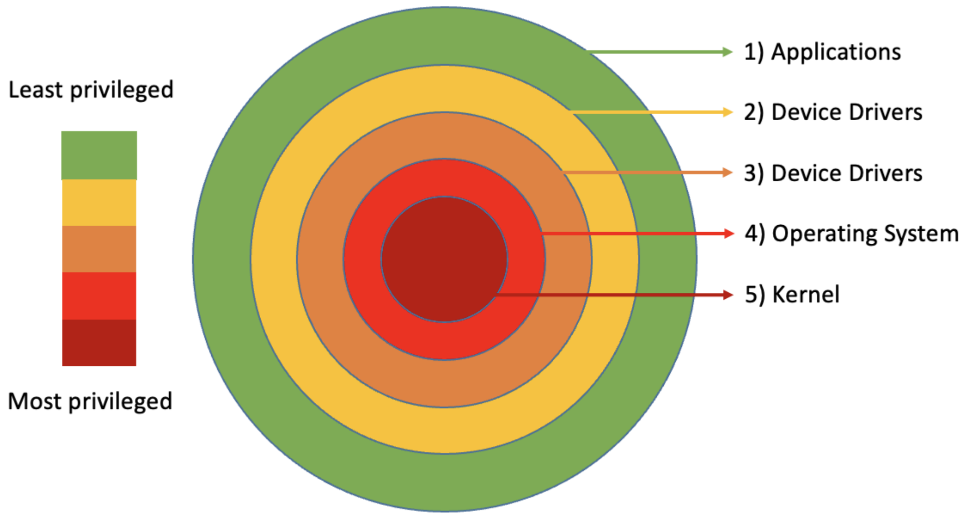


Figure 4.2: Privilege Ring

In our use case definition, section 4.4, we choose Intel SGX to develop and explore its properties in a cloud environment. In normal situations in cloud, the data are saved altogether and the system administrators can “access” them and confidentiality of the user data of cloud services is only ensured by the trust in the cloud services provider. It means that it cannot be guaranteed that a cloud service provider company will not access private user data that was stored in the cloud or that aggressors will find loopholes that can hack the system. There may also be the problem of an insider abuse authority threat. But not only this, imagine if someone is able to gain access to the Cloud Service Provider it will be a total mess. He can basically do everything with the data or even more, and in this case the integrity, and confidentiality of data is broken. In our case, we will consider three types of aggressors. First, the normal user who has only limited access to data may aim to attack the cloud provider. The second is a foreign user who has no access to any of the services and the third is a user from inside the cloud. The last user is because he may have access to the system files, so he can have an advantage on understanding the code and trying to break it. Example 1. A user wants to attack a hypervisor on the database VM server, to steal all information running on that machine. He starts by getting the initial access to the database VM by which he will attack the hypervisor and by exploiting CVE-2013-4344, the attacked can get data related to all VMs or use the hypervisor to get access to the target VM. This may lead to data loss, etc.

Using TEE, we have the coexistence of trusted and untrusted applications in the same embedded device, which they can communicate only if they trust each other. This is crucial because the goal of a TEE is to separate the important code and functionalities from the environment.

In general, the system cannot trust the HW and OS unless this is configured in the TEE, and which functions to trust. The OS can get corrupted and try to get/modify our secrets. Since the goal is to provide secure Data Isolation, information flow in the trusted environment is quite limited. OS cannot add/modify code in the TEE, its functions are limited towards the secure environment such as, only to secure its secrets. In order for an aggressor to find the vulnerabilities, the following preconditions are required, but not limited:

Network embedded systems: The majority of embedded systems are connected to the internet, which makes them a perfect target for the aggressors. No physical access is needed to exploit the vulnerabilities and break the security mechanism.

Weak Encryption algorithm: Based on the encryption algorithm used, the data may be stolen and decryption methods such as brute-forcing may be used to break the security. This can happen because the algorithm uses a key with a small length, which is below 128 bits [43].

Weak Software and Hardware architecture: In some cases, there are software security measurements to secure the data, and the attacker can break the security easier. Also, hardware isolation solutions have their own drawbacks when it comes to security, as we will see AMD SEV does not provide data integrity, even though it provides TEE. Sometimes vulnerabilities on the software can risk also the HW technology, even though they are not supposed to impact each other. For example.

4.3. Threat Model

We believe that the cloud service provider will maintain all the security parameters and the data will not be shared with third parties unless explicitly said in the contract. But the cloud providers for many reasons cannot be fully trusted, there may be malicious employees, etc., which may be interested in breaking and compromising the confidentiality and integrity of hosted containers.

Users are given roles, which based on this they can access different data. This role is given by the main user. A normal user cannot access data if not specified, to protect from possible malicious users, if one manages to request data his role will be checked inside the enclave.

We do not trust the Linux Kernel, firmware, hypervisor, or the running programs on the cloud, but we only trust the Trusted execution environment inside the CPU. We create a code partition of trusted and untrusted functions. This may be time-consuming, but it should be done, so aggressors cannot run a malicious which can compromise the enclave.

We assume that each user has a two-factor authentication to protect from loss/theft of their credentials. We assume that the attacker cannot have physical access to the cloud, so we do not consider different physical attacks.

Intel SGX will play a critical role on:

1. Allowing application developers to protect sensitive data from malicious software running at higher privilege levels, trying to gain unauthorized access.
2. Preserve the confidentiality and integrity of sensitive code and data and at the same time system software can schedule and manage the use of platform resources.
3. Inside Enclaves we will be generating Key Pairs for each user, Encrypting/Decrypting data, and creating trusted functions for our application to run.

4.4. Use Case

A hospital used an old CRM system to manage their clients, we can assume that the servers used to be based on the hospital, or in another location, but still managed by the hospital's IT staff. This method is an old solution, it can be costly, hard to maintain, and easy to target from aggressors since the technology mostly will be the same for some years. The hospital can suffer from many cyber security issues including malware that can be taken from emails or any peripheral device which will lead to failure risks, data breaches, etc. There are a lot of benefits to using the cloud and enforcing security through hardware means. Figure **??**represents a high-level architecture of our cloud solution, including different actors and entities that take place.

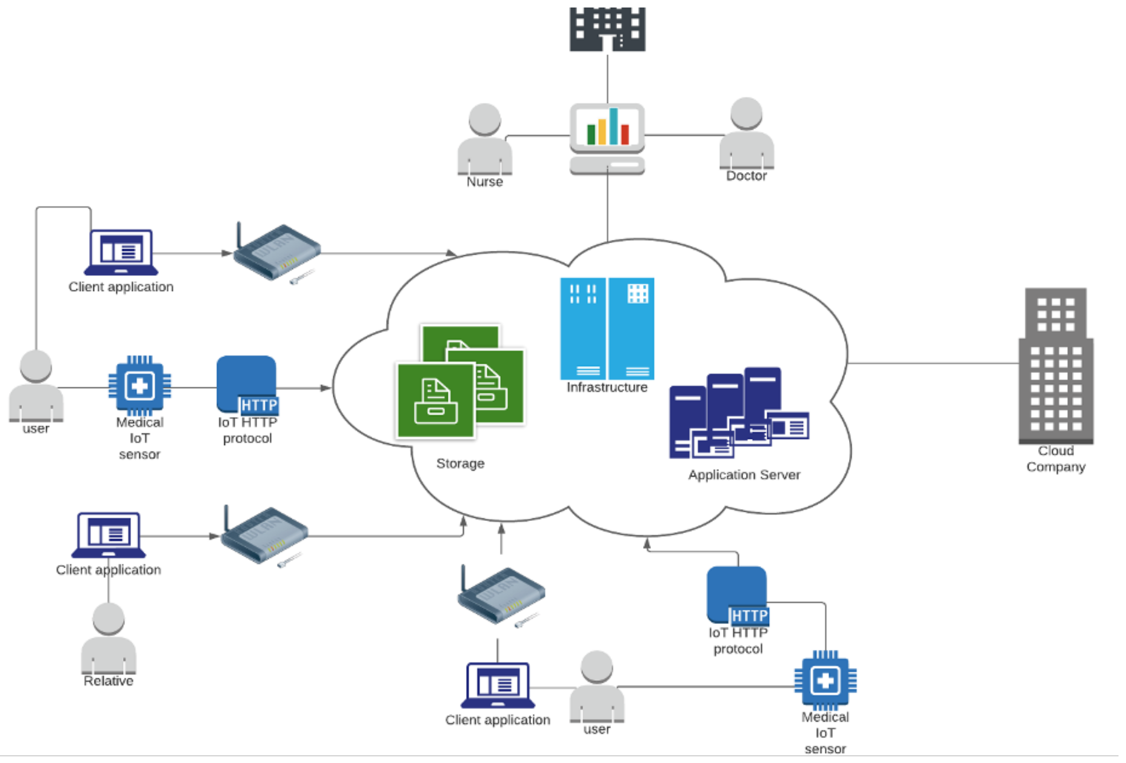


Figure 4.3: High-level representation of our Cloud Architecture shows the architecture of our cloud, different actors, and devices all connected and streaming or retrieving data online.

Actors

In this section, we describe what are the different actors that compose our use case! These actors are the ones who will be "using" the system through the client application. For this use case, it is important, from a security point of view, to make some assumptions regarding some possible threats that can affect our system which is not the scope of this thesis!

Actors	Roles
Relatives	Basic
Patient	Basic
Nurse	Normal
Doctors	High
Managers	Admin

Table 4.1: List of actors and their role/privilege on our system

Relatives:

1. Basic Role
2. Can only view and control the data of their relatives. It can modify some basic functionalities.

Patients:

1. Basic Role
2. Provides the healthcare data from IoT device
3. Cannot see and control anything but its own data

Nurse:

1. Can insert/update/delete data for users
2. Only of those they have under their care
3. Can register new users
4. Can view the data of hospital patients

Doctors:

1. Can Insert/update/delete users' data
2. Can register new users
3. Can view data of hospital patients
4. Can assign Nurses to users

Manager:

1. Can Insert/delete/update users' data
2. Can Edit/register Doctors etc.

Devices:

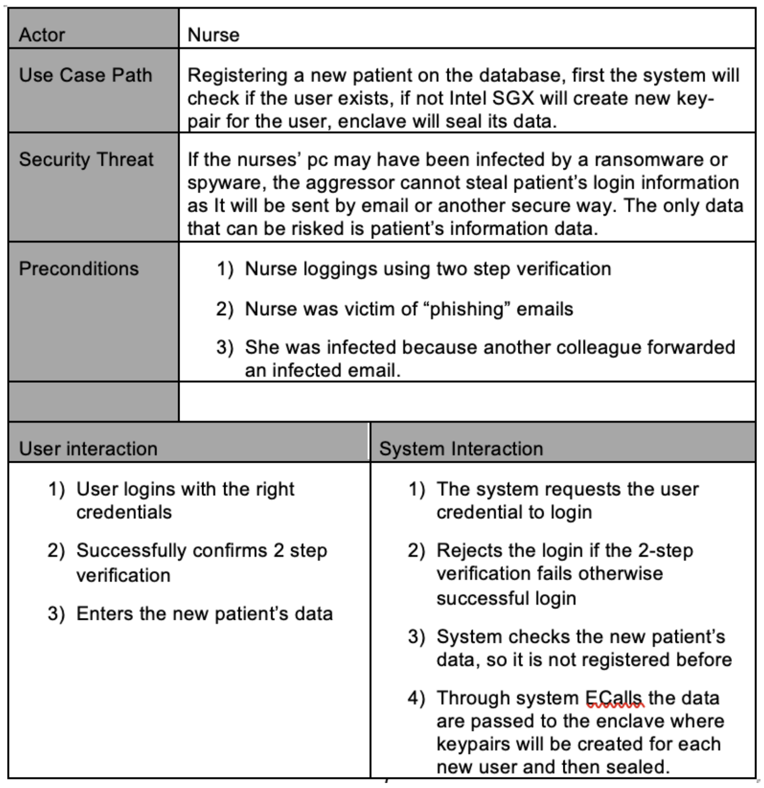
1. IoT health sensor: A device that is connected and registered to a patient and will send data to the cloud in a secure manner.

4.5. Scenarios

In this section we are going to explain some scenarios and sequence diagrams for better understanding the functions and data flow in the system, and why it is necessary to take the right measurements. First let's describe some of the main entities and what are their roles on this case.

Scenario 1

In this scenario, a nurse will be registering a new patient from the client app and all the business logic will be described.



Actor	Nurse
Use Case Path	Registering a new patient on the database, first the system will check if the user exists, if not Intel SGX will create new key-pair for the user, enclave will seal its data.
Security Threat	If the nurses' pc may have been infected by a ransomware or spyware, the aggressor cannot steal patient's login information as It will be sent by email or another secure way. The only data that can be risked is patient's information data.
Preconditions	1) Nurse loggings using two step verification 2) Nurse was victim of "phishing" emails 3) She was infected because another colleague forwarded an infected email.

User interaction	System Interaction
1) User logins with the right credentials 2) Successfully confirms 2 step verification 3) Enters the new patient's data	1) The system requests the user credential to login 2) Rejects the login if the 2-step verification fails otherwise successful login 3) System checks the new patient's data, so it is not registered before 4) Through system ECalls the data are passed to the enclave where keypairs will be created for each new user and then sealed.

Figure 4.4: First Scenario of the use case, registering a new user on the database

To enforce the first scenario, figure 4.5 describes a sequence diagram showing different group objects communicating with each other to provide security for the data.

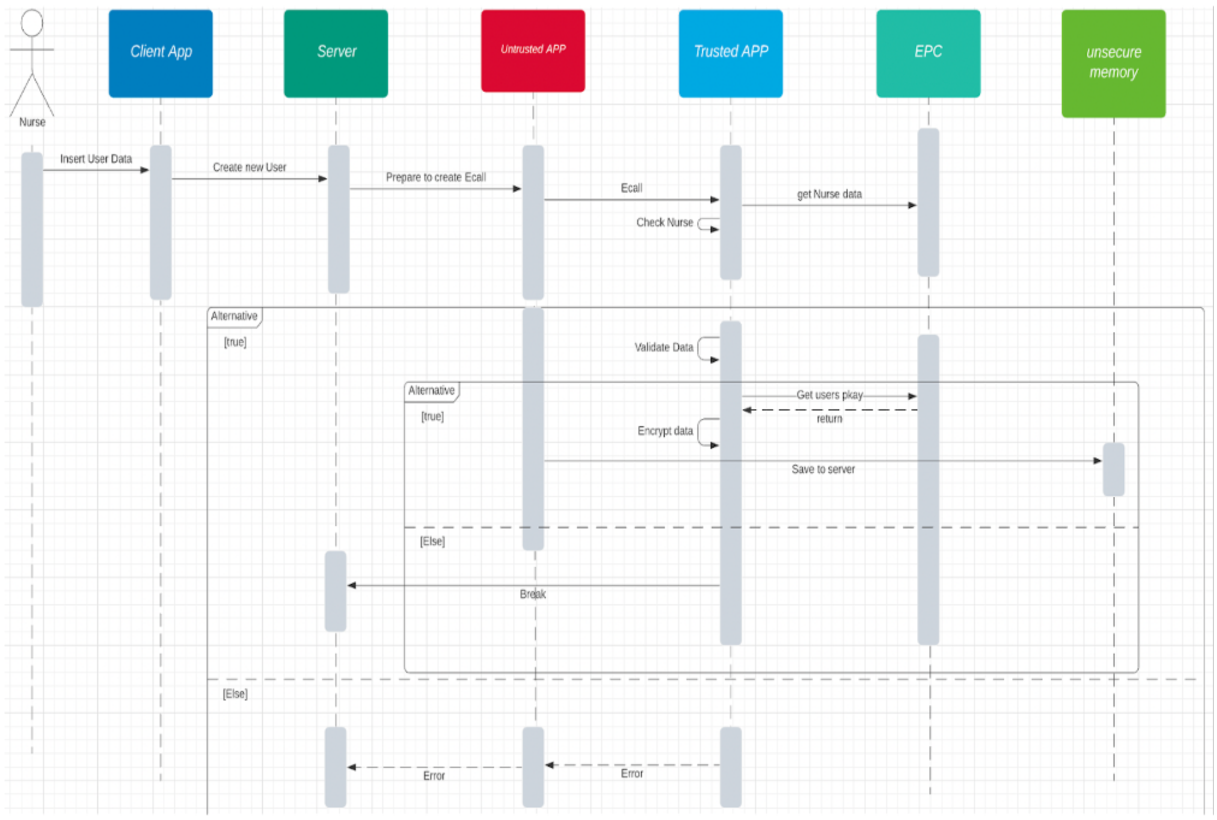


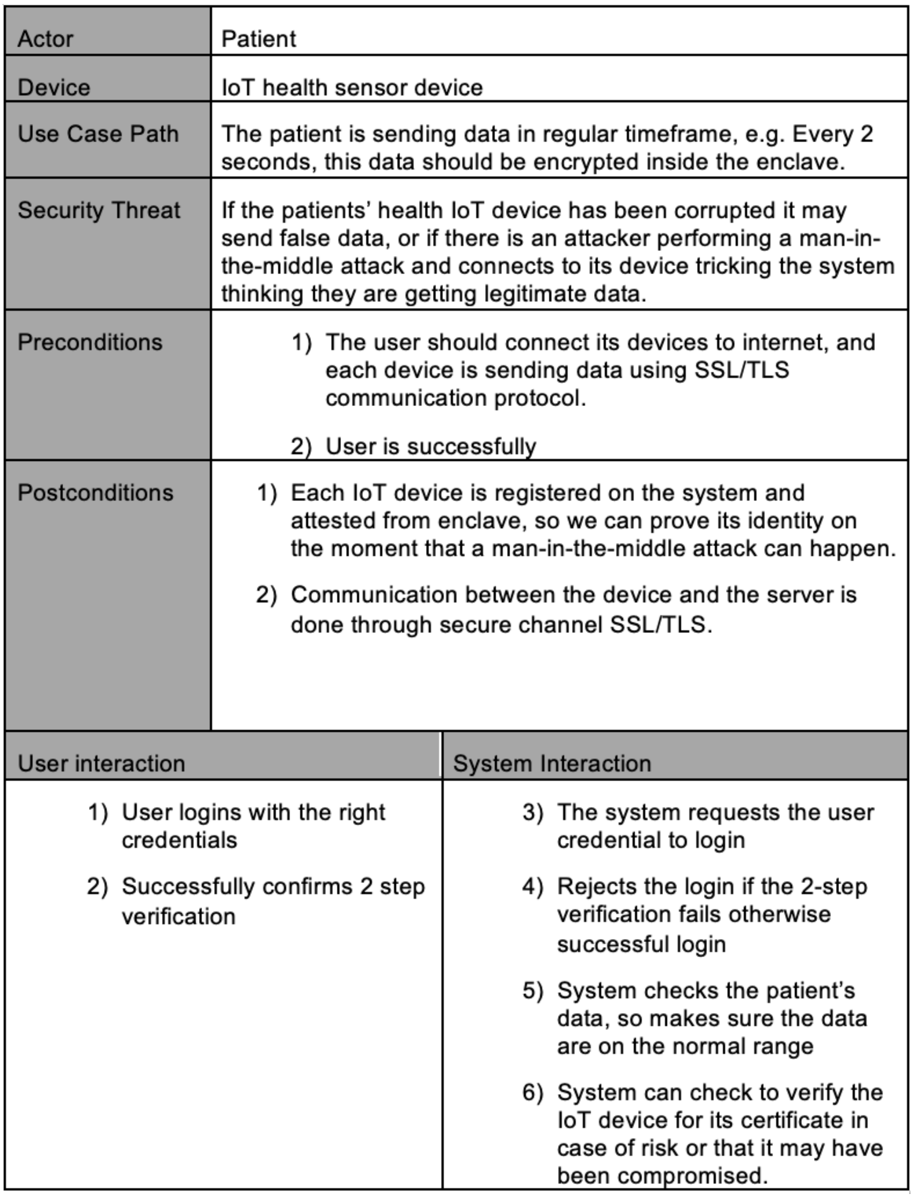
Figure 4.5: Nurse Registering a new Patient, Intel SGX validates nurse, and saves users data to unsecure memory after encryption inside the Enclave

Scenario 2

In this scenario user is sending will be registering a new patient from the client app and all the business logic will be described.

1. Before saving the data controller if proof of identity.
2. Check if data are valid and are not suspicious.
3. Encrypt the data with the public key and store them into the main memory.

The communication channel between the IoT device and the Cloud is secured using SSL. This way we believe that the data won't be corrupted in transit, but still, there is a need to check them. If we take a scenario where we blindly trust the data from the Device, the proposed Sequence Diagram makes Sense, otherwise, we need to add a control scheme for the data which I propose below.



Actor	Patient
Device	IoT health sensor device
Use Case Path	The patient is sending data in regular timeframe, e.g. Every 2 seconds, this data should be encrypted inside the enclave.
Security Threat	If the patients' health IoT device has been corrupted it may send false data, or if there is an attacker performing a man-in-the-middle attack and connects to its device tricking the system thinking they are getting legitimate data.
Preconditions	1) The user should connect its devices to internet, and each device is sending data using SSL/TLS communication protocol. 2) User is successfully
Postconditions	1) Each IoT device is registered on the system and attested from enclave, so we can prove its identity on the moment that a man-in-the-middle attack can happen. 2) Communication between the device and the server is done through secure channel SSL/TLS.

User interaction	System Interaction
1) User logins with the right credentials 2) Successfully confirms 2 step verification	3) The system requests the user credential to login 4) Rejects the login if the 2-step verification fails otherwise successful login 5) System checks the patient's data, so makes sure the data are on the normal range 6) System can check to verify the IoT device for its certificate in case of risk or that it may have been compromised.

Figure 4.6: Second Scenario, the patient is sending data to the cloud database

To enforce the first scenario, figure 4.7 describes a sequence diagram showing different group objects communicating with each other to provide security for the data. On the first Alternative check, the system controls if the user is not a malicious attacker, and inside enclave on the trusted app, we check for its authorization.

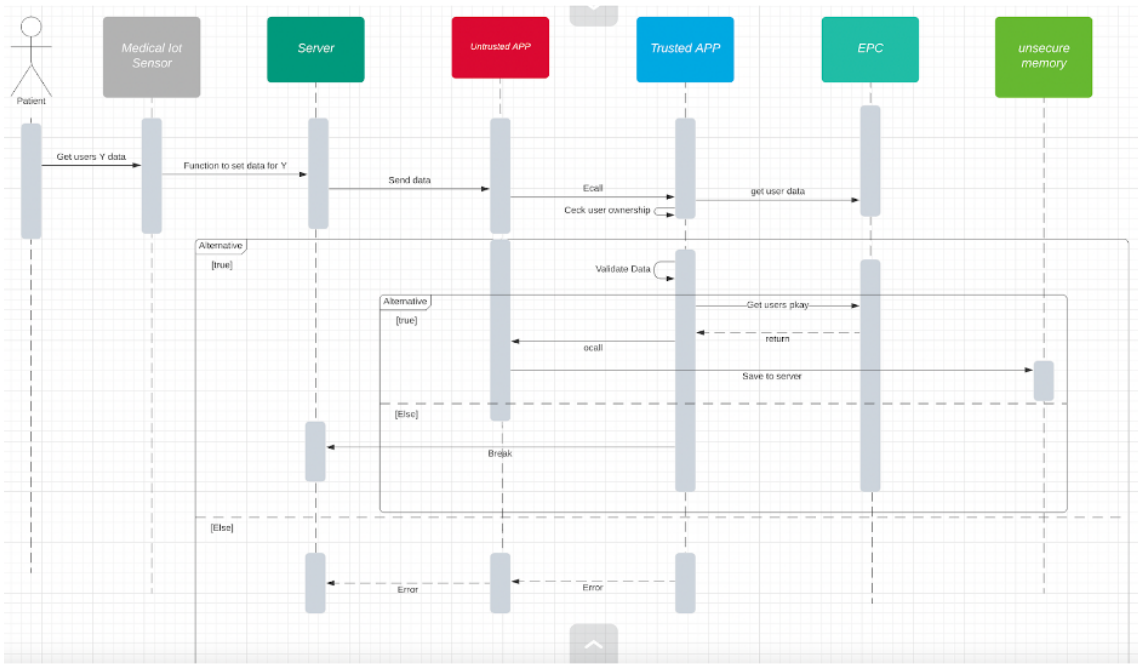


Figure 4.7: Sequence diagram of patient sending data to the cloud.

4.6. Technical Background

In this section, we are going to talk about the different security principles and concepts used for our use case. Some of the concepts are also described in Section 3 when we talk about Intel SGX and its features. The section will start by giving some important information on how we plan to use Intel SGX on our use case, the use of confidentiality, integrity, attestation, cryptographic solutions, and the basic concepts.

The experiment will be done on a pc that comes with Intel SGX capability in CPU and BIOS and the OS is Ubuntu 20.04. Intel SGX should be enabled on BIOS so we can install the drivers and the SDK. SGX will create the trusted executed environment by setting aside a memory region called the Processor Reserved memory. CPU protects PRM from all non-enclave memory accesses, including the kernel, hypervisor, etc. PRM is important because it will hold all the Enclave Page Cache (EPC), which consists of 4KB pages that store enclave code and data. This secure data cannot be accessed by untrusted entities, as mentioned in section 2.5, but only by the enclave that created them. An important element that we want to secure is the encryption key pairs for each user. As the data will be stored in the server, the encryption keys need a secure environment. So even if it is the case that someone can gain user-level access, and maybe reach for the data he won't be able to decrypt them. This is important for public cloud storage or hybrid storage because the data are stored in a shared infrastructure can be a target for many attackers.

So, we want to provide confidentiality, the integrity of data, and code integrity.

Architecture Solution

In this section, we study more about our architecture and how Intel ® SGX will help us to reach our scope as mentioned in the previous section.

Assumptions:

- We assume that the user who can access the data from the client application is not an aggressor and his account is not compromised.
  - Our goal is not to prove user authentication from the client application point of view but Intel SGX helps us to protect his data so we can provide data confidentiality.
- The connection between the IoT device to the cloud is secured.
  - We assume that the Device uses one of the secure communication protocols SSL/TLS to minimize data leaking and man-in-the-middle attacks. [19]
- Data Recovery is provided.

The assets which we need to protect are

1. the private key
2. the user data

The data after the secure encryption inside Intel SGX can be saved inside the unsecure memory in the cloud, but the private key which is used for encryption is saved in the EPC.

Application Design:

To build the application design we need to clarify some points which will be useful in designing the enclaves.

1. Identify the secrets
2. Providers and Consumers Of secrets
3. Determine Enclave Boundary
4. Tailor the Code

To program inside the enclave, we need to use the Enclave Definition Language syntax. Both ECALL and OCALL are prototyped inside the EDLs. Intel SGX doesn't not call the ECALL and OCALL functions directly; it calls the proxy functions. On the moment that of an ECALL, first is called the untrusted proxy function for the ECALL, which in turn calls the trusted proxy function inside the enclave. That proxy then calls the "real" ECALL and the return value propagates back to the untrusted function. This sequence is shown in Figure 22. When the system makes an OCALL, the sequence is reversed: first call the trusted proxy function for the OCALL, which calls an untrusted proxy function outside the enclave that, in turn, invokes the "real" OCALL.

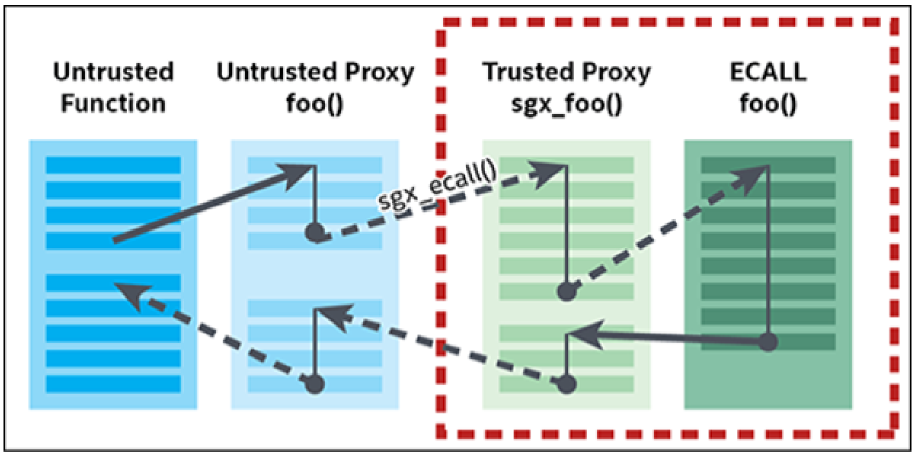


Figure 4.8: API communication between trusted and untrusted domains. sd From: Intel® Software Guard Extensions Part 7: Refine the Enclave with Proxy Functions, 2016 [4]

```
enclave {
        // Include files
        // Import other edl files
        // Data structure declarations to be used as
        //parameters of the function prototypes in edl

        trusted {
        // Include file if any. It will be inserted
        // in the trusted header file (enclave\_t.h)
        // Trusted function prototypes (ECALLs)
        };

        untrusted {
        // Include file if any. It will be inserted in
        // the untrusted header file (enclave\_u.h)
        // Untrusted function prototypes (OCALLs)
        };
};
```

Identify the secrets

Secrets are the data that should not be known or seen by other users. Only the users' owners of this data can edit, view delete his secrets. In no way the secrets should be exposed to other users or applications regardless of their privilege level. For example, in figure **??** we show a sequence diagram of the data flow from an untrusted environment, the application checks for the ownership before passing the data, get the private key from EPC, performs decryption Inside the enclave on TEE, and sends the data.

For our use case secrets can include the following data:

1. Medical records
2. Healthcare data from IoT devices
3. Encryption keys
4. Enclave keys

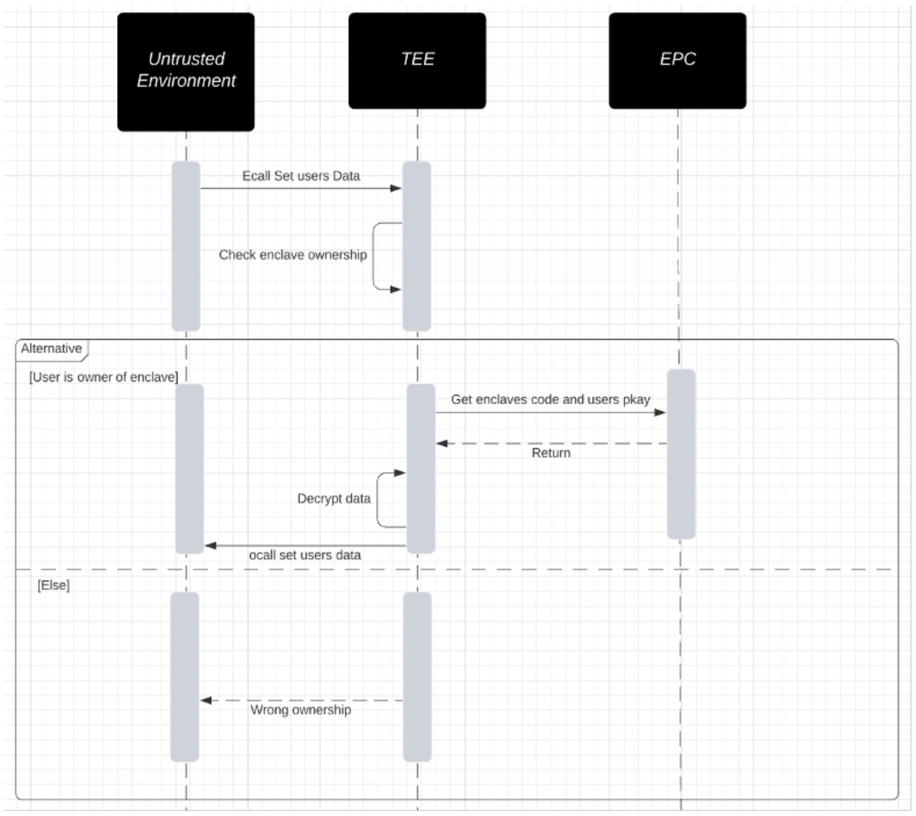


Figure 4.9: High level sequence diagram of data flow between domains.

Data flowing from the untrusted environment to the EPC

It is important that the data going to the untrusted environment to be secure. Because it is the only place where there is no security, and the data can be manipulated. SGX design supports having multiple enclaves on a system at the same time, which is a necessity in multi-process environments. This is achieved by having the EPC split into 4 KB pages that can be assigned to different enclaves. The EPC uses the same page size as the architecture's address translation feature. The EPC is managed by the same system software that manages the rest of the computer's physical memory. The system software, which can be a hypervisor or an OS kernel, uses SGX instructions to allocate unused pages to enclaves and to free previously allocated EPC pages. Non-enclave software cannot directly access the EPC, as it is contained in the PRM. Inside the TEE (trusted execution environment) is where all the trusted code will be located to perform our crucial functions. This protects the code from being accessed from outside. Some of the main functions which will go here are:

Identify the providers & consumers of secrets

When we create the secrets, they can enter or leave the enclave, but the most important is to minimize them to untrusted code. Intel SGX uses the enclave to perform the encryption or decryption of our data. One important aspect will be Sealing and Attestation (Remote Attestation). This is because at the moment that our data will be encrypted, we may lose them since the enclave life cycle ends by destroying itself. So, every data cannot be decrypted, thus we can derive an encryption key that can be used to provision encrypted secrets to the application that only the trusted enclave on that client system can decrypt.

As mentioned, the data can travel outside the enclave to the untrusted code, the important part is that they are encrypted all the time.

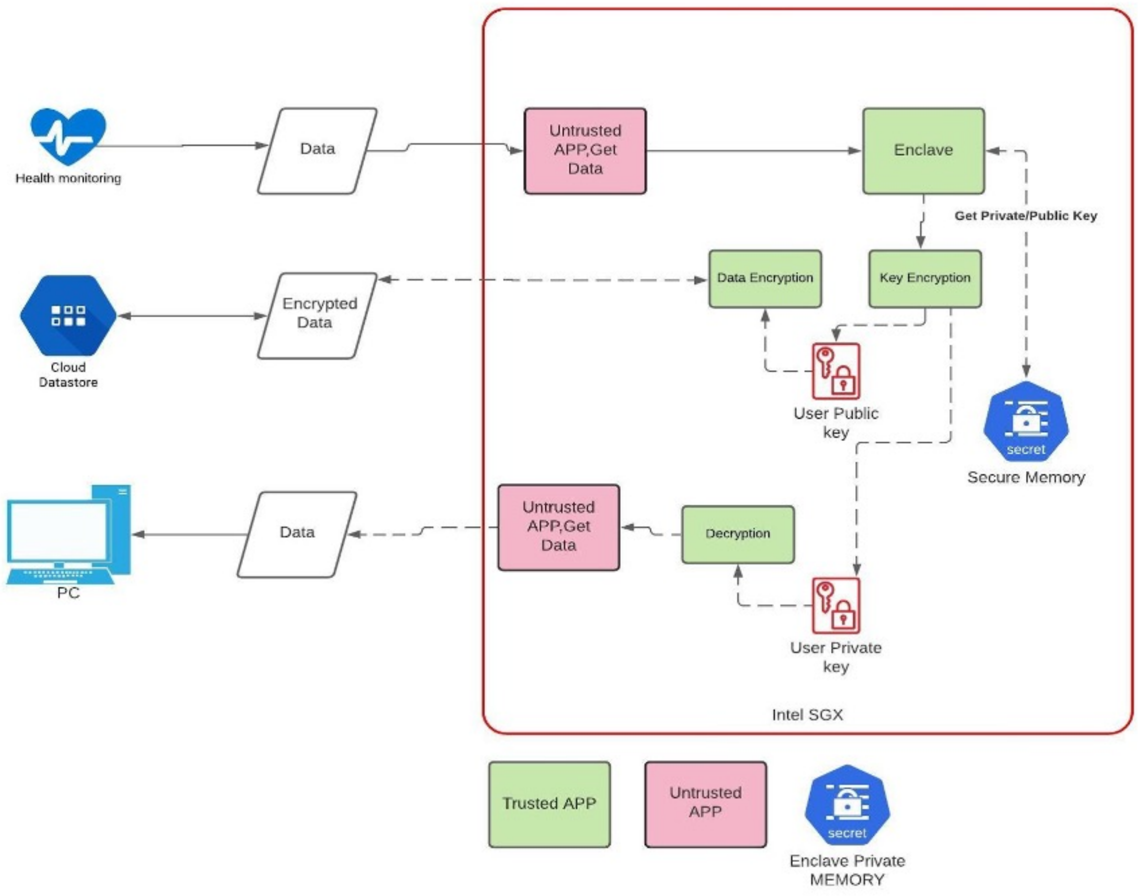


Figure 4.10: Data Flow of different entities to the cloud and Intel SGX functionalities

Figure 4.10 represents a data flow, where we have different providers of our secrets and different consumers. The main provider is the IoT device which will provide health monitoring data, after a secure communication with the backend data are redirected to Enclave for secure encryption which will be saved on the cloud database. When data come from the device, using secure communication protocol SSL/TLS, they need to be encrypted before being saved inside the Cloud Database. Here comes the use of Intel SGX, by using the Crypto Library we generate a new pair of keys so they can be used from Intel SGX to encrypt and decrypt the data. The private key is sealed and saved inside the secure memory and the public key is saved to the Cloud database.

Determine Enclave Boundary It is important to understand the main data flow through the application's core components. By this, the enclave boundary should give an understanding of how the different components act with our data. It should provide minimum access to the secrets and the components that act upon them. Minimize the interaction with the untrusted code unless needed.

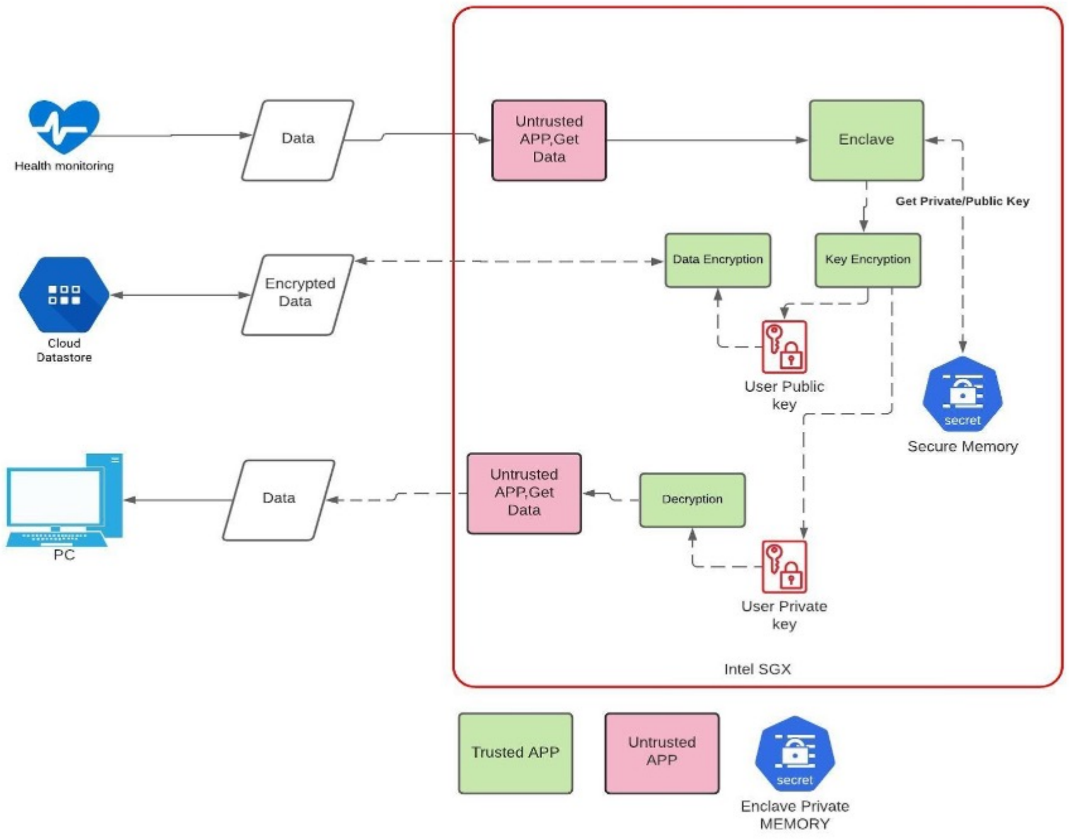


Figure 4.11: Data flow from different Entities, to the Intel SGX

The user's private-public key is generated inside the enclave and not outside. Then it is encrypted with the enclave key and saved securely. The enclave's private key is saved inside the EPM and never send outside the enclave otherwise the whole idea of a trusted execution environment is gone. We can have a problem with the way our data will be shown to the user. Since the data is taken from the database and then the decryption is done inside the enclave with the user's private key, when it goes to the untrusted environment it will be on plain text. And idea is to use OpenSLL so the data will have minimum encryption when they are on transit. When the data is outside the enclave, it should never be saved in any variable or whatsoever, since some libraries have their own memory management it is hard to control all the data saved everywhere for no reason.

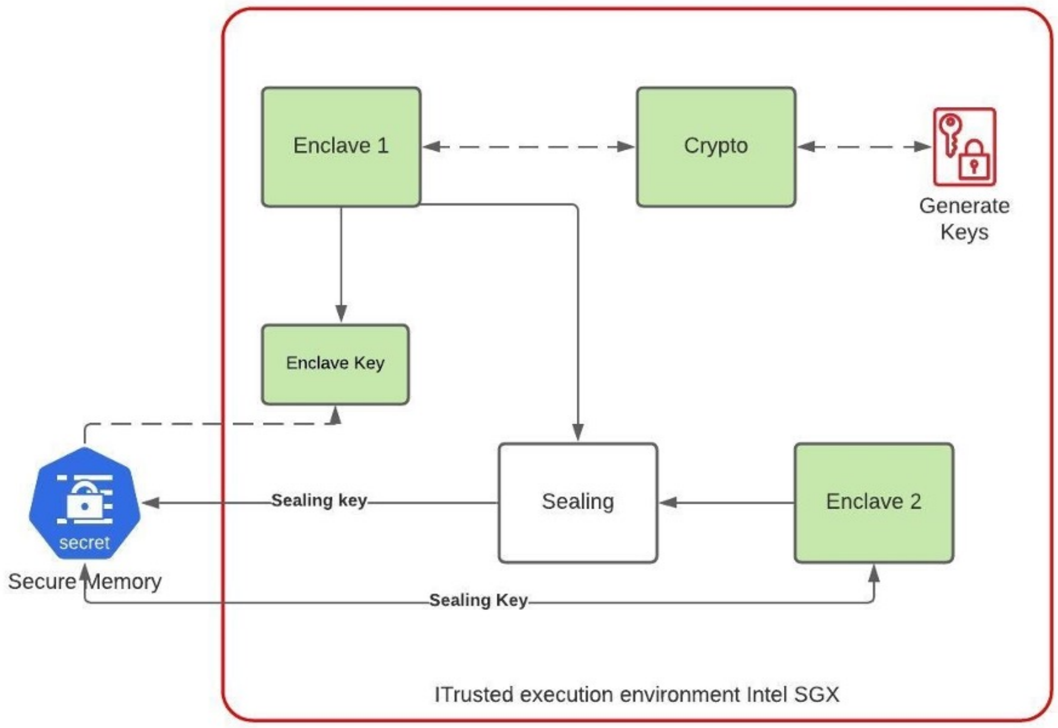


Figure 4.12: Sealing and generating encryption keys

In this diagram it is shown how the keys are generated and the use of Sealing. Sealing it is important because it guarantees that the data will be encrypted when the Enclave is restored otherwise, they will be lost. The Crypto function creates the enclaves' keys to be used and is saved in secure memory. Which cannot be accessed by outside users and nor by untrusted code.

Code Tailoring

On the specifications of Intel SGX, it is recommended to tailor the code in such a way that it will minimize the risks and the trusted functions are well distinguished from the untrusted ones. For our Use Case we want to focus on the creation of new users, generating key pairs, attesting the user and the IoT device and sealing their data.

Classes that I am going to use for the trusted and untrusted domains: Trusted App: These functions are called inside the trusted execution environment and run inside the enclave. Some of these functions may need to use oCall and eCall APIs to get or send data from an untrusted environment to the trusted execution environment.

Ecall_check_user

Before creating a new user, we need to make sure that this user profile does not exist, in case the user has the same name and family name as another user, the person registering will need to give a confirmation.

Ecall_Create_User_File
This class will create an encrypted file, which will serve as the “database” for the user. The reason why we are using this approach is for simplicity, but this file plays the same role as a real database. The data inside will be sealed and encrypted using users and enclave private keys.

Ecall_key_generation_and_sealing
When a new user gets registered, this function will be called inside to generate two key pairs for him and seal this data so they will be secured inside the secure memory. This key pair will be later used to encrypt and decrypt users’ data inside the enclave.

Ecall_show_item
This function will be used to decrypt the data of the user from the database. When someone requests to get the data for x user the data need to be decrypted inside the enclave then can be shown to the user. For security measurements, we cannot send the keys outside a trusted execution environment.

Ecall_add_data
As shown in scenario 2, the user will be sending data to the database periodically. This function inserts the new data on the cloud database. First, the data will be encrypted, sealed, and then saved.

ecall_remove_data
On the moment that someone decides to delete data from the database, we must delete the key pairs of the user. This function takes care of it. Untrusted App: These functions are called within the enclave to exit enclave temporarily and call a function in the untrusted space.

Ocall_show_info
At the moment we call the function Ecall_show_item, for security reasons Enclave cannot be used to send the data to the user client. What Intel SGX provides is the Ocall library, where you call a function from the untrusted environment inside the Enclave to perform a certain task. In this case, our function will take the decrypted data from the enclave,

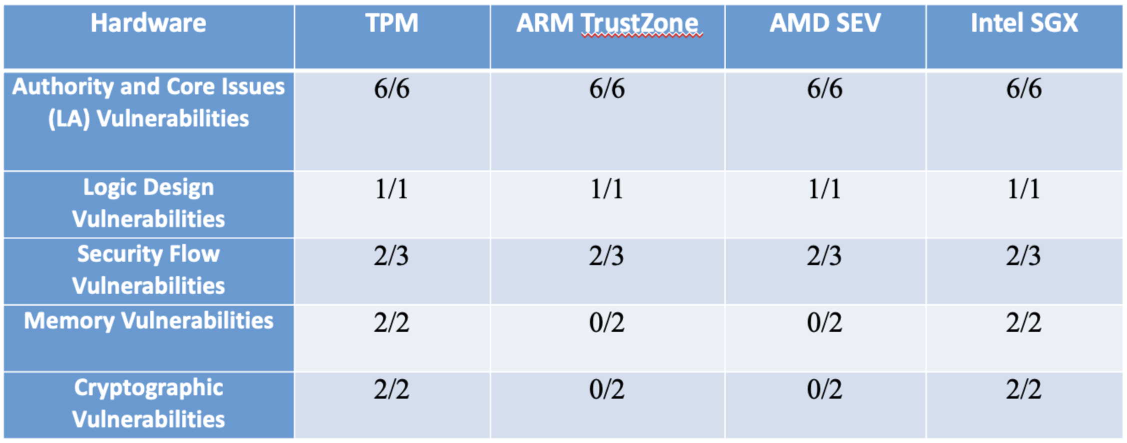
pass it through API in a secure manner to the untrusted environment. As mentioned on Section 2.5, the data inside the enclave can be accessed only if the owner is proven.

Ocall_save_file
The functions inside the trusted environment Ecall_Create_User_File and Ecall_add_data decrypt the data sent by user and then call the function Ocall_save_file to pass the data and save them. Ocall_user_exists
Ecall_check_user will check if the user exists on the database and send a return message to the untrusted environment through the Ocall_user_exists.

5 | Conclusion

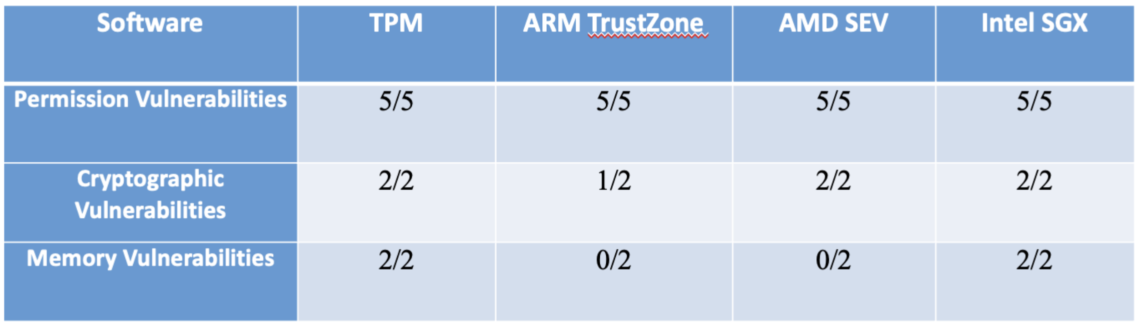
5.1.

This work explored Trusted Execution technologies by differentiating their commons and differences regarding the HW and SW component, how they protect or not w.r.t different vulnerabilities. A list of different selected vulnerabilities were analyzed, to see how they impact these technologies, which components were affected the most and which can provide better protection. Intel SGX was the main target of this work, and so it was analyzed more in details. From this work we can conclude that Intel SGX provides a good and novel protection mechanism, which comes with the cost of some drawbacks.



Hardware	TPM	ARM TrustZone	AMD SEV	Intel SGX
Authority and Core Issues (LA) Vulnerabilities	6/6	6/6	6/6	6/6
Logic Design Vulnerabilities	1/1	1/1	1/1	1/1
Security Flow Vulnerabilities	2/3	2/3	2/3	2/3
Memory Vulnerabilities	2/2	0/2	0/2	2/2
Cryptographic Vulnerabilities	2/2	0/2	0/2	2/2

Figure 5.1: This image represent the total number, n/m, of 5 different HW CWE vulnerability categories, that each TEE technology can protect from. Where m is the total nr. of vulnerabilities, and n is the nr. they can protect from.



Software	TPM	ARM TrustZone	AMD SEV	Intel SGX
Permission Vulnerabilities	5/5	5/5	5/5	5/5
Cryptographic Vulnerabilities	2/2	1/2	2/2	2/2
Memory Vulnerabilities	2/2	0/2	0/2	2/2

Figure 5.2: This image represent the total number, n/m, of 3 different SW CWE vulnerability categories, that each TEE technology can protect from. Where m is the total nr. of vulnerabilities, and n is the nr. they can protect from.

From the results on the Figure 5.2 and Figure 5.1 we conclude that Intel SGX and TPM can provide a better secure environment w.r.t CWE vulnerabilities.

Regarding the main architecture elements for SW and HW part, the only difference between the four technologies is that ARM TrustZone does not provide sealing, Integrity and attestation is missing for both ARM TZ and AMD SEV. AMD SEV uses VM to create the secure executed environment. ARM TZ by default does not provide encryption techniques, so its need some SW or HW crypto accelerators.

In our use case when the data were stored in a unsecure memory, database, but they were all encrypted. So even if an attacker can manage to get the data from database, he wont be able to decrypt them, as the algorithm that can be used, provides good encryption keys which will take basically forever to brute-force decryption. Second, the way how enclaves works, even if the user manages to get inside an enclave, which is almost impossible and really hard without physical access to the device, it cannot get access to all the data and cannot risk integrity and confidentiality. From the research done Intel SGX can provide a better secure environment than the other technologies. On one hand, enclave provides some mechanism to provide attestation, in order to "trust" third parties. In our case it the idea to be used to attest IoT devices. Intel SGX provides two types of attestation, Intel® Enhanced Privacy ID (Intel® EPID) Attestation and Elliptic Curve Digital Signature Algorithm (ECDSA) Attestation. Differences between this two are mentioned on section 2.5.1. We used EPID, because of technology limitations, ECDSA usually is available only on Intel server processors. On the other hand, it wasn't possible to reach all the wanted goals. We used Linux Ubuntu 20.04 as OS and to set the working environment. There were many limitations on the developer documentations of Intel SGX for Ubuntu.

5.2. Future work & improvements

The outcome of this work can be better improved by using an Intel server processor in order to explore the full capabilities of Intel SGX. On section 2.5.1 we show the differences between two types of attestation that Intel SGX provides,ECDSA and EPID, but they cannot both be applied on normal Intel processors. ECDSA requires XEON processors while EPID can run on any normal Intel processors who support Intel SGX. Another interesting approach can be using another computation protection SW, which can interact with Intel SGX to run functionalities which need protection inside the Enclaves. These technologies can be combined to perform a better secure environment. An example is Ferraiuolo Andrew 2017 [22], which decouple the core HW and SW mechanisms to work together. Some other research that can be done, is to measure the time delay of the encryption and decryption process till the moment that data are provided the the user, which can be a critical part for environments with very high density of data.

Bibliography

[1] Trustzone technology for armv8-m architecture version 2.1, . URL https://developer.arm.com/documentation/100690/latest/.

[2] The trustzone hardware architecture, . URL https://developer.arm.com/documentation/100935/0100/The-TrustZone-hardware-architecture-.

[3] Linux " linux kernel : Vulnerability statistics. URL https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33.

[4] Intel® software guard extensions part 7: Refine the enclave with..., Oct 2016. URL https://www.intel.com/content/www/us/en/developer/articles/training/intel-software-guard-extensions-tutorial-part-7-refining-the-enclave.html.

[5] Nov 2017. URL https://github.com/intel/intel-sgx-ssl/blob/master/Intel(R)SoftwareGuardExtensionsSSLLibraryArchitecture.pdf.

[6] https://www.amd.com/system/files/techdocs/sev-snp-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, Jan 2020. URL https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[7] Demystifying arm trustzone for microcontrollers, Sep 2020. URL https://developer.arm.com/documentation/dui0446/u/debugging-embedded-systems/about-accessing-ahb--apb--and-axi-buses.

[8] September 2020 healthcare data breach report: 9.7 million records compromised, Oct 2020. URL https://www.hipaajournal.com/september-2020-healthcare-data-breach-report-9-7-million-records-compromised/.

[9] Business technology reports, 2020. URL https://www.verizon.com/business/resources/reports/.

[10] Cve-2021-33909, Jun 2021. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-33909`.

[11] 2021. URL `https://developer.arm.com/documentation/PRD29-GENC-009492/c/TrustZone-Hardware-Architecture`.

[12] Cve-2021-33909, Nov 2021. URL `https://access.redhat.com/security/cve/cve-2021-33909`.

[13] R. N. Akram, K. Markantonakis, and K. Mayes. *An Introduction to the Trusted Platform and Mobile Trusted Module*, pages 71 – 94. 09 2013. ISBN 978-1-4614-7914-7. doi: 10.1007/978-1-4614-7915-4_4.

[14] G. Arfaoui, S. Gharout, and J. Traoré. Trusted execution environments: A look under the hood. In *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 259–266, 2014. doi: 10.1109/MobileCloud.2014.47.

[15] J. Arnold, T. Abbott, W. Daher, G. Price, N. Elhage, G. Thomas, and A. Kaseorg. Security impact ratings considered harmful. *arXiv preprint arXiv:0904.4058*, 2009.

[16] W. Arthur, D. Challener, and K. Goldman. *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security.* Springer Nature, 2015.

[17] Biya and Ar. 30 examples of embedded systems in daily life, Aug 2019. URL `https://compscistation.com/examples-embedded-systems-daily-life/`.

[18] E. Brickell and J. Li. Enhanced privacy id from bilinear pairing for hardware authentication and attestation. volume 1, pages 768–775, 08 2010. doi: 10.1109/SocialCom.2010.118.

[19] Z. Cekerevac, Z. Dvorak, L. Prigoda, and P. Čekerevac. Internet of things and the man-in-the-middle attacks – security and economic risks. *MEST Journal*, 5:15–5, 07 2017. doi: 10.12709/mest.05.05.02.03.

[20] H. Chen, Y. Mao, X. Wang, D. Zhou, N. Zeldovich, and M. F. Kaashoek. Linux kernel vulnerabilities: State-of-the-art defenses and open problems. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, pages 1–5, 2011.

[21] V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016 (86):1–118, 2016.

[22] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 287–305, New York,

NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350853. doi: 10.1145/3132747.3132782. URL `https://doi.org/10.1145/3132747.3132782`.

[23] M. J. Haber. Privilege escalation attack and defense explained, Oct 2021. URL `https://www.beyondtrust.com/blog/entry/privilege-escalation-attack-defense-explained`.

[24] S. Han, W. Shin, J.-H. Park, and H. Kim. A bad dream: Subverting trusted platform module while you are sleeping. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1229–1246, 2018.

[25] M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono. On technical security issues in cloud computing. In *2009 IEEE International Conference on Cloud Computing*, pages 109–116, 2009. doi: 10.1109/CLOUD.2009.60.

[26] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[27] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[28] R. Kunkel, D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer. Tensorscone: A secure tensorflow framework using intel sgx. *arXiv preprint arXiv:1902.04413*, 2019.

[29] K. Kursawe, D. Schellekens, and B. Preneel. Analyzing trusted platform communication (2005).

[30] B. W. Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

[31] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[32] S. E. Madnick and J. J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proceedings of the workshop on virtual computer systems*, pages 210–224, 1973.

[33] Y. Mao, H. Chen, D. Zhou, X. Wang, N. Zeldovich, and M. F. Kaashoek. Software fault isolation with api integrity and multi-principal modules. In *Proceedings of the*

Twenty-Third ACM Symposium on Operating Systems Principles, pages 115–128, 2011.

[34] D. Moghimi, B. Sunar, T. Eisenbarth, and N. Heninger. Tpm-fail:{TPM} meets timing and lattice attacks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2057–2073, 2020.

[35] M. A. Mukhtar, M. K. Bhatti, and G. Gogniat. Architectures for security: A comparative analysis of hardware security features in intel sgx and arm trustzone. In *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*, pages 299–304, 2019. doi: 10.1109/C-CODE.2019.8680982.

[36] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin. Trustzone explained: Architectural features and use cases. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 445–451. IEEE, 2016.

[37] A. Nilsson, P. N. Bideh, and J. Brorsson. A survey of published attacks on intel sgx. *arXiv preprint arXiv:2006.13598*, 2020.

[38] R. P. Pires. Distributed systems and trusted execution environments: Trade-offs and challenges. *CoRR*, abs/2001.09670, 2020. URL `https://arxiv.org/abs/2001.09670`.

[39] J. Regehr and U. Duongsaa. Preventing interrupt overload. *SIGPLAN Not.*, 40 (7):50–58, jun 2005. ISSN 0362-1340. doi: 10.1145/1070891.1065918. URL `https://doi.org/10.1145/1070891.1065918`.

[40] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 2015.

[41] A.-R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy. Tcg inside? a note on tpm specification compliance. In *Proceedings of the first ACM workshop on Scalable trusted computing*, pages 47–56, 2006.

[42] V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski. Supporting third party attestation for intel sgx with intel data center attestation primitives. *White paper*, 2018.

[43] C. Sincerbox. Security sessions: Exploring weak ciphers - an explanation and an example, 2014. URL `https://electricenergyonline.com/energy/magazine/779/article/Security-Sessions-Exploring-Weak-Ciphers.htm`.

[44] D. Suciu, S. McLaughlin, L. Simon, and R. Sion. Horizontal privilege escalation in trusted applications. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[45] G. Tan et al. *Principles and implementation techniques of software-based fault isolation*. Now Publishers, 2017.

[46] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 991–1008, 2018.

[47] Windows-Driver-Content. Trusted plaform module (tpm) 2.0, May 2021. URL `https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-tpm`.

[48] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou. Truspy: Cache side-channel information leakage from the secure world on arm devices. *IACR Cryptol. ePrint Arch.*, 2016:980, 2016.

[49] R. Zhang, R. Xue, and L. Liu. Security and privacy on blockchain. *ACM Computing Surveys (CSUR)*, 52(3):1–34, 2019.

[50] Y. Zhang, Y. Sun, Y. Sun, R. Jin, K. Lin, K. Lin, W. Liu, and W. Liu. High-performance isolation computing technology for smart iot healthcare in cloud environments. *IEEE Internet of Things Journal*, pages 1–1, 2021. doi: 10.1109/JIOT.2021.3051742.