Institute of Information Security
Universitätsstraße 38
70569 Stuttgart

Bachelorarbeit

Comparison of Prominent Trusted Execution Environments

Xiaoyu Zhang

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Ralf Küster
Supervisor:	Dr. Daniel Rausch
Commenced:	September 17, 2021
Completed:	March 17, 2022

Abstract

In recent years, the need for people to be able to do anything on the go has risen. Especially in eastern countries like China, India, and Japan online payment methods like WeChat pay, Alipay, or other mobile wallets are becoming more predominant, making old-fashioned cash transactions obsolete. This trend has led to rising security requirements for applications running on smartphones or other mobile devices. Therefore, the devices must be able to process transactions for the user and service providers confidentially. A solution to this are Trusted Execution Environment (TEE), which provide an isolated execution environment and secure storage where users can store and process vital information, for example, passwords, biometrics, or cryptographic primitives. Two prime solutions are Intel SGX, developed by Intel and included in most Intel processors. The other one is ARM TrustZone, used in the processors of many mobile devices like smartphones or Internet of Things (IoT) devices, examples include the chips for smartphones produced by Qualcomm.
This new approach was developed because the system software was becoming increasingly unreliable in the past few years. Because of the large code size of common operating system (OS) like Windows and Android, no one could guarantee that there were no exploits or other attack vectors that could be abused by malicious parties. This can be seen by the number of security updates for these systems. Another problem is that the user of the system is not necessarily trustworthy either, and he might use it to steal information from other parties, for example, copyrighted content. For this reason, TEEs were developed, because of their small code-base they are less vulnerable to attacks, as the attack surface is reduced and more manageable. Additionally, it is able to keep secrets from the OS and the user, enabling more use cases that were previously only possible on the server side. For example, microtransactions where authentication requires sensitive input from the user to more complex ones like verifiable cloud computing where vital computations are executed by potentially untrustworthy third parties.
This thesis aims to compare two prominent TEE, Intel SGX, and ARM TrustZone, in two aspects, how they perform against common security attacks and how they perform in common use cases. Common security attacks contain expensive physical attacks to more sophisticated cache timing attacks. The use cases discussed in depth include digital rights management, anonymous attestation, secure multiparty computation, and verifiable cloud computing among others.

Kurzfassung

In den letzten Jahren hat das Bedürfnis der Menschen zugenommen, alles unterwegs erledigen zu können. Vor allem in asiatischen Ländern wie China, Indien und Japan setzen sich online-Zahlungsmethoden wie Wechat Pay, Paypay oder andere mobile wallets immer mehr durch und machen altmodische Bargeldtransaktionen obsolet. Dieser Trend hat zu steigenden Sicherheitsanforderungen für Anwendungen auf Smartphones oder anderen mobilen Geräten geführt, die in der Lage sein müssen, Geheimnisse von Nutzern und Dienstleistern vertraulich zu verarbeiten. Dies führte zu der Enticklung von TEE. Sie bieten eine isolated execution environment und einen secure storage, in dem die Nutzer wichtige Informationen wie Passwörter, biometrische Daten oder kryptografische Primitive speichern und verarbeiten können. Zwei bekannte Beispiele dieser Technologie sind Intel SGX, das von Intel entwickelt wurde und in den meisten Intel-Prozessoren enthalten ist. Das andere Beispiel ist ARM TrustZone, eine Architektur, die in den Prozessoren vieler mobiler Geräte wie Smartphones oder IoT-Geräte verwendet wird, z. B. in den von Qualcomm hergestellten Chips für Smartphones.

Dieser neue Ansatz wurde entwickelt, weil die Systemsoftware in den letzten Jahren immer unzuverlässiger wurde. Aufgrund der großen Codegröße der meisten Betriebssysteme wie Windows und Android konnte niemand garantieren, dass es keine Sicherheitslücken oder andere Angriffsmöglichkeiten gab, die von Angreifern missbraucht werden konnten, was an der Menge der Sicherheitsupdates für diese Systeme zu sehen ist. Ein weiteres Problem besteht darin, dass der Benutzer dieses Systems nicht unbedingt vertrauenswürdig ist und es dazu benutzen könnte, Informationen von anderen Parteien zu stehlen. Beispiele dafür sind urheberrechtlich geschützte Inhalte. TEE bieten eine Lösung für diese Probleme. Sie sind aufgrund ihrer kleinen Code-Basis weniger anfällig für Angriffe, da die Angriffsfläche kleiner und überschaubarer ist. Darüber hinaus können sie Geheimnisse vor der OS und dem Benutzer bewahren, was mehr Anwendungsfälle ermöglicht, die bisher nur auf der Serverseite möglich waren. Zu diesen Anwendungsfällen gehören häufigere Anwendungen wie mobile transaction, bei denen die Authentifizierung sensible Eingaben des Benutzers erfordert, bis hin zu komplexeren Anwendungen wie verfiable cloud computing, bei dem wichtige Berechnungen von potenziell nicht vertrauenswürdigen Dritten ausgeführt werden.

In dieser Arbeit werden zwei prominente TEE Technologien vorgestellt, nämlich Intel SGX und ARM TrustZone. Diese werden unter zwei Aspekten verglichen, zum einen wie sie sich gegen gängige Sicherheitsangriffe verhalten und zum anderen wie sie in gängigen Anwendungsfällen funktionieren. Gängige Sicherheitsangriffe umfassen teure physical attacks bis hin zu ausgefeilteren side-channel attacks. Zu den Anwendungsfällen, die in dieser Arbeit eingehend erörtert werden, gehören unter anderem die digital rights management, anonymous attestation, secure multiparty computation und verfiable cloud computing.

Contents

List of Figures

Acronyms

BLXNS	branch and link with exchange to non-secure state. 28
BXNS	branch with exchange to non-secure state. 28
CPU	central processing unit. 15
DMA	Direct Memory Access. 49
Dos	Denial of Service. 31
DRAM	Dynamic random access memory. 16
DRM	Digital Rights Management. 13
ELRANGE	Enclave Linear Address Range. 33
EPC	Enclave Page Cache. 32
EPCM	Enclave Page Cache Map. 32
EPID	Enhanced Privacy ID. 24
GB	Giga Byte. 18
GDXC	Generic Debug eXternal Connection. 47
GIC	Generic Interrupt Controller. 31
I/O	Input/Output. 15
IoT	Internet of Things. 3
KB	Kilo Byte. 18
MB	Mega Byte. 18
NS	Non-Secure. 27
OS	operating system. 3
PRM	Processor Reserved Memory. 32
SCR	Secure Configuration Register. 27
SECS	SGX Enclave Control Structure. 32
SG	Secure Gateway. 28
SMC	Secure Monitor Call. 28
SMM	System Management Mode. 43

SSA	State Save Area. 33
TEE	Trusted Execution Environment. 3
TLB	translation look-aside buffer. 18
TPM	Trusted platform module. 13
TZASC	TrustZone Address Space Controller. 30
TZMA	TrustZone Memory Adapter. 30
TZPC	TrustZone Protection Controller. 31

1 Introduction

In recent years, the demand of people to be able to do anything on the go has risen. One example is a mobile wallet, an extension of the credit card system where applications on smartphones can be used to pay and receive money. Especially in eastern countries like China, India, and Japan, this payment method is widespread, with applications like WeChat pay, AliPay, or other mobile wallets being prominent examples. In the year 2020 2.2 billion people were using mobile payment applications worldwide, more than 5 times the amount of users than in the year 2015 [1]. And it is not the only application on user devices that handle high security information, with Digital Rights Management (DRM) and mobile identity being other prominent use cases that require high security standards, something that the average system software is not able to provide nowadays. This is because OS like Android and Windows have extremely large codebases, e.g. Windows 11 requiring at least 64 gigabyte of memory. This large codebase means a very large attack surface for malicious parties who could examine it for potential vulnerabilities that could be exploited for attacks. And the amount of vulnerabilities in system software is very high, which can be seen by the large number of security updates OS like Windows and Android receive. This inevitably leads to the OS being unable to guarantee the integrity of the software and the confidentiality of anything stored in it. Therefore, traditional system software is considered untrustable, unable to provide the security requirements necessary to handle confidential information like copyrighted content or other important data.
A solution to this problem was developed by the trusted computing group in 2009, and it is called the Trusted platform module (TPM). It provides a secure hardware area in a computer that could be used to store important information, ranging from cryptographic keys to copyrighted content, which would be protected from a potentially malicious OS. Another use case for the TPM was to act as a root of trust for the device it was part of, verifying that the device was behaving as intended. One key feature that was missing in the TPM was to provide an execution environment that could be used to run important applications, severely hampering its usability for certain use cases. Because of this, the TEE was developed. In addition to a secure storage provided by the TPM, it also included an isolated execution environment that could be used by applications for higher security standards. Two prominent examples of this technology are ARM TrustZone and Intel SGX. ARM TrustZone is a hardware TEE technology developed by ARM and included inside most of its processors. These processors are widely used on smartphones and IoT devices, with famous examples including the chips produced by Qualcomm which are used in most Android smartphones. Intel SGX is another hardware TEE technology developed by Intel and is more commonly found in larger computers like laptops or servers, as these devices predominantly use Intel processors. This thesis aims to compare these two TEEs from two aspects: protection again common security threats and usability in common use cases.
The security threats compared in this thesis can be summarized into five categories. The first category are the physical attacks, these attacks include non-invasive ones such as power analysis attacks that aim to extract information about the current calculation by observing the power consumption of the processor and semi-invasive attacks like chip imaging attacks. The second category are

the privileged software attacks, these include any attacks that use higher privilege levels to attack applications with lower ones. This category is one of the main concerns of TEE. Another category of attacks discussed in this thesis are software attacks on peripherals. This type of attack uses peripheral devices to gain access to memory or enable other attacks. The fourth category are the address translation attacks, these attacks include passive ones where the memory access pattern of applications is monitored to active ones that aim to cause malfunctions by manipulating the address translation results. Another form of attack are cache timing attacks, these attacks abuse the side effect of caches in that caches respond faster if the memory access is already allocated, whereas if it is not then the response time is slower as the memory has to get loaded first. This leads to attackers being able to infer patterns which in turn can be used to extract important data, such as cryptographic keys.

The common use cases discussed in this thesis include digital rights management. These involve any applications that are used to protect and distribute copyrighted contents such as movies, songs, or games and are one of the more prominent use cases involving TEE. Another prominent use case in this thesis, especially on mobile devices like smartphones, are mobile payments methods and mobile identity, where the authentication and subsequent payment involve sensitive information about the user that have to be protected. Another use case that is explored is the use of TEE to detect malware or even protect against it. Since the TEE is isolated from the system software an infected system does not imply the TEE has also been compromised, as a consequence it can be used to check for abnormal behaviour and serve as a root of trust. A more advanced use cases discussed in this thesis are anonymous attestation, a way for the TEE to prove its integrity to third parties anonymously. Secure multiparty computation is another use case where multiple participants execute a piece of code using inputs from all parties while not revealing the inputs. The final use case shown in this thesis is verifiable cloud computing, as the cloud providers are not necessarily trustworthy, a TEE can be used to ensure the integrity and confidentiality of the execution happening in the untrusted cloud.

This thesis continues with Chapter 2, where the topics required to understand the rest of the thesis are presented. Chapter 3 discusses the implementation of the TEE to be compared in the thesis. The thesis proceeds with Chapter 4, where the two environments will be compared in regards to common security threats. Chapter 5 compares their performances when applied to common uses cases. Chapter 6 concludes the thesis.

2 Foundation

This chapter presents the required knowledge to understand the functionality of TEE. Section 2.1 will introduce the basics of computer architecture. It covers topics such as how a central processing unit (CPU) works and memory management. Section 2.2 will further delve into the topic of TEE and define it in the scope of this thesis.

2.1 Computer Architecture

This section summarizes the general principles of computer architecture. The first subsection gives a general overview of common computer architectures in the form of a simplified computational model. The second subsection goes over the basics of processors, including software privilege levels and how they execute instructions. The final subsection goes over memory management consisting of the different types of memories and how they are accessed.

2.1.1 Overview

This subsection presents a simplified version of the von Neumann architecture, an early computer architecture described by John von Neumann in 1945 [2]. The functionality of the computer since its inception in the early 20th century can be summarized as being devices that are used by humans to execute functions. These functions range from simple calculations in calculators or early computers to more complex modern applications like web servers or operating systems. No matter at which point in time or on what kind of device, every computer relies on three core components to execute functions, these being the logical processor, a device to store data, and Input/Output (I/O) devices through which the outside world can influence or start these functions. These components are connected via the system bus and are managed by the OS. This simplified model is depicted in Figure 2.1.

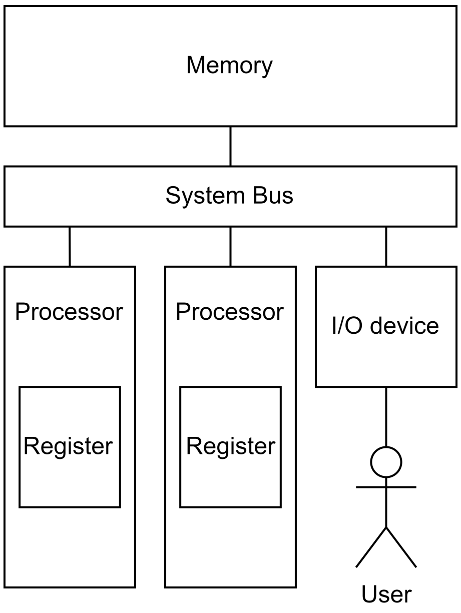

Figure 2.1: computer model

The I/O devices, also known as peripherals, are responsible for the interaction of the computer with the outside world, in the picture represented by the common user. It can also be sensors or other computers which detect changes and alert the processors accordingly. Through these I/O devices, it is possible to give instructions for the computer to execute which are stored in memory. Memory can take many forms, for example, larger memory like the Dynamic random access memory (DRAM) or smaller but more responsive memory in the form of register files and caches. No matter which form memory takes in the computer they are all fundamentally storage cells that can be accessed using natural numbers. The processors can then read the instructions stored in memory, execute them, and write the results of the execution into memory again, which can then be outputted through a respective I/O device. It should be mentioned here that a traditional von Neumann architecture separates the von Neumann architecture into two different units, with one being responsible for instructions and the other being used for arithmetic logic. They are referred to as the control unit and the logic unit respectively. This is not represented here as it is not relevant to this thesis.

2.1.2 Processors

Processors are the main component of computers, as they enable the execution of instructions. These instructions are bundled together in a process, as singular instructions only enable the use of basic operations such as read, write, add, and subtract. Only together do they allow for more complex actions such as the calculation of complex functions. Modern computers support the running of multiple processes, which are supported by multiple processors managed by the OS. To achieve this the OS allocates memory and processors to the different processes and isolates them using virtualization. Virtualization can be summarized as creating multiple virtual computers by using software to simulate hardware functionality on the same device [3]. This enables the execution of multiple applications on the same device, which provides better scalability and efficiency than earlier solutions. Larger computers, for example, servers, benefit from this technology. Another advantage is that it prevents malicious processes from negatively interfering with other processes. Additionally, it reduces software complexity as developers do not need to keep interactions with other processes in mind. Address Translation is a key part of virtualization technology. It gives the processes an illusion that they own a very large amount of storage, maybe even all available storage. This component will be further described in Section 2.1.3. Software privilege level or protection ring is another part of virtualization technology. Software have different levels of access to resources, with more privileged levels having access to all resources available at less privileged levels. This partitioning of resources protects higher level rings from faults or malicious code of lower levels. The x86 architecture, the most common computer architecture, defines four levels of privilege, starting at ring 0 for the kernel and ending at ring 3 for applications. These rings can be seen in Figure 2.2

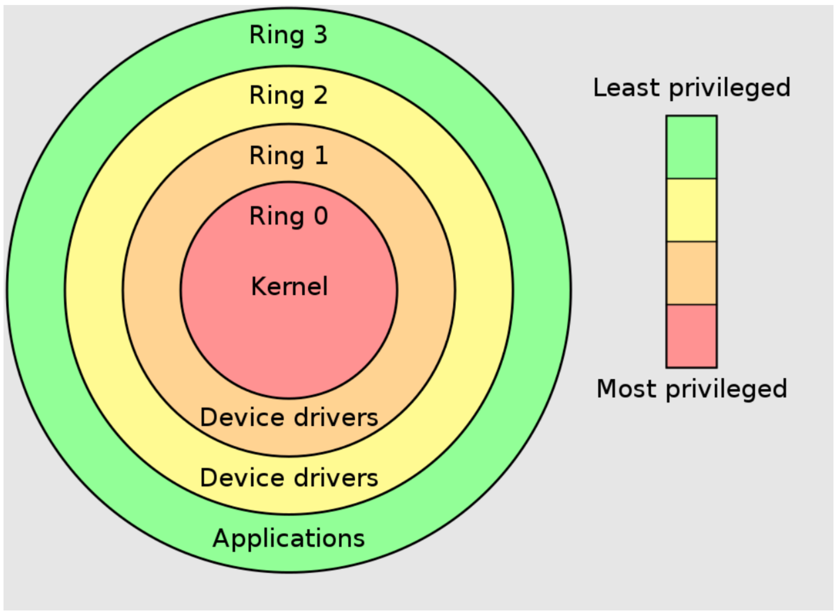

Figure 2.2: software privilege levels in an x86 architecture [4]

Processes are executed by creating multiple threads. These threads are a sequence of instructions that are executed by a logical processor. This starts with the assignment of threads to processors, which is managed by the OS and can happen concurrently or sequentially. Because of virtualization processors have the impression that they have access to an infinite amount of logical processors. The number of threads spawned by this impression from different processes is managed using different methods such as preemptive multi threading, which will not be further covered in this thesis. Finally, the corresponding logical processors execute the instructions provided by the threads. It should be mentioned that modern processors can execute instructions much faster than they can be loaded from DRAM. Accordingly, modern architectures provide smaller but faster memory for the processors, such as register files and caches. After a process is computed the results are stored in memory from where they can then be outputted to the user via I/O devices.

2.1.3 Memory Management

The second main component of computers is memory. Memory can take many forms, with each differing in size and access speed. The first form of memory related to the scope of this thesis is the DRAM. Compared to hard drives it is fast and because of its affordable price relatively large, ranging from 8 Giga Byte (GB) on flagship smartphones to 32 GB on high end computers. Processor caches are another form of memory, and it is faster than DRAM. These are further divided into different levels, ranging from Level 1 to Level 3. They are up to 100 times faster than DRAM. However, because of their higher cost, they are mostly limited between 32 Kilo Byte (KB) for Level 1 caches up to 32 Mega Byte (MB) for Level 3 caches. The fastest form of memory available to processors are the registers, they are integrated into the processor itself and are consequently more expensive than caches. For this reason, they are only available in a very limited amount.
The different forms of memories also get different address spaces, specified by the respective developers. For example, Intel uses four address spaces, these being register spaces, memory spaces, I/O spaces, and model-specific register space. Register memory is addressed using register spaces, which are defined by CPU architecture. These registers are further divided into different categories, with control registers being responsible for the operation of the CPU and therefore only accessible by the OS. The rest of the registers can be used by all applications at all software privilege levels. The memory space contains the addresses of memory mapped devices, such as the DRAM. This memory space is further partitioned by the OS for the different connected devices. The I/O space is usually referred to as ports and is responsible for communicating with the outside world. The allocation of this space is managed by different standards. The model-specific register space consists of the memory responsible for the operation of the CPU. Consequently, any instructions that interact with these memory addresses can only be performed by the OS to prevent faulty behaviour.
Because of the small size of faster memory, they only hold the most often used portions of a processes instructions. The rest are fetched from larger memory when they are needed. This creates the illusion that the memory is much larger than it actually is, and this illusion is enabled by address translation and called virtual memory [5]. Address translation maps the larger virtual memory to smaller physical addresses and this happens at the level of pages. The physical memory is therefore partitioned into different pages, each with its own number with the virtual memory referencing the respective pages. The basic functionality of this system can be summarized as follows. When virtual memory is addressed, the OS will first check whether the translated address is already cached in the translation look-aside buffer (TLB). If it is, the physical address will be taken from the cache. If not, then the virtual address will be translated to the actual physical address. If the addressed virtual memory is loaded in the physical memory then it could be instantly accessed by the processor. If virtual memory is addressed that has not been loaded into the actual physical memory, the virtual page will be paged in. In case the physical memory is full, the least used page will be paged out in the progress. This requires the OS to be aware of the actual location of the memory, if it is aware of it this piece of memory is considered mapped. An unmapped page is memory that is deallocated and its mapping information is not available. Another important aspect is called virtual address aliasing, this means that the same physical page may be referenced by multiple virtual addresses. This among other example mappings can be seen in Figure 2.3. Here the physical page 3 is pointed at by the virtual addresses 0x00003 and 0xFFFFE.

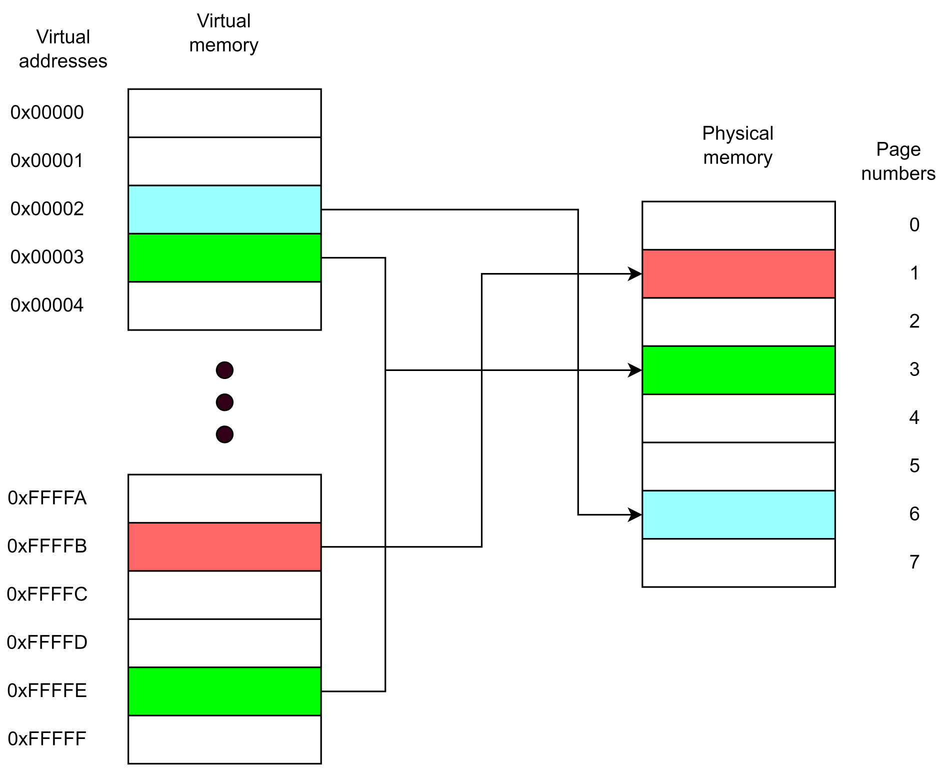

Figure 2.3: address translation example

2.2 Trusted Execution Environments

This section gives a brief overview of TEE. The first subsection presents the basics of a TEE, what it is and how it is commonly used. The second subsection describes one way of how TEE achieves isolated execution, an important feature of all TEEs. The third subsection shows how TEEs protect their memory from malicious software, referred to as secure storage. The remaining subsections go over other desirable features of TEEs, these being remote attestation, secure provisioning, and trusted path. The final subsection concludes with common Trusted Execution Environments and their core features.

2.2.1 Trusted Execution Environments Basics

With the ever increasing demand for security, traditional technologies are no longer sufficient. Trusted Computing was therefore introduced to provide secure computation, privacy, and data protection. Initially realized on separate hardware, called TPM, this solution soon became insufficient. This is because while the TPM provided evidence of its integrity and could securely store cryptographic keys on its separate module, it could not provide a tamper resistant environment for the execution of code [6]. Another solution for secure computation was the TEE, examples of which include Intel SGX and ARM TrustZone. In contrast to TPM a TEE provides an isolated execution environment that allows for the execution of code without malicious software interfering with it, from here on out referred to as Isolated Execution which will be discussed further in Section 2.2.2. Another core feature of TEE is its ability to provide secure storage for confidential information, for example, cryptographic keys, this will be further described in Section 2.2.3. Other desirable features for trusted computing bases in addition to the previous two were defined by Vasudevan in his paper about trustworthy execution on mobile devices [7]. These include remote attestation, which is the ability of third parties to verify the integrity of a TEE it wants to use. This feature is further described in Section 2.2.4. Another feature is called secure provisioning, which is the ability to send data to the TEE while protecting it from the potential malicious device it is part of. Section 2.2.5 describes this in more detail. The final feature defined by Vasudevan is called trusted path. It ensures the authenticity of the communication happening between the TEE and a peripheral, preventing attackers from spoofing peripherals to gain access to the environment. This feature will be further discussed in Section 2.2.6. All these features are desirable in trusted execution environments on common computers such as laptops or smartphones, but they may not necessarily be supported. Table 2.1 gives an overview of these features.

Table 2.1: desirable features of trusted execution environments

Feature	Description
isolated execution	running code isolated from surrounding code like system software, preserving the integrity of the executed code
secure storage	store data in a secure area, for example a privileged memory area or a separate storage device, protects the integrity of the data
remote attestation	enable third parties to verify the integrity of the TEE in question, establishing trust between both parties
secure provisioning	enable a secure way to transfer data between the TEE and the third party and protect its integrity
trusted path	a path between the TEE and a peripheral that protects the authenticity of it, preventing malicious spoofing attempts

2.2.2 Isolated Execution

One core feature of any TEE is isolated execution. This means that sensitive code can be executed separately from potentially compromised OS. Isolated execution is commonly achieved by having the sensitive code execute in a secure mode, sometimes on a physically protected part, referred to as secure world in TrustZone and as enclave mode in Intel SGX. The secure mode is implemented differently for every TEE technology. While some have the processor execute the code on the same processor with extra security features, others execute it on a physically different processor that can only be accessed in secure mode and is invisible normally. This has led to several new definitions of additional software privilege levels below level 0, which was previously the most privileged level in the x86 architecture. One definition comes from Ning, Zhang, and Shi in their paper about hardware assisted execution environments which defines three additional levels more privileged than level 0 [8]. Privilege level -1 is reserved for hypervisors, which are executed at the same level as the kernel on level 0 in other models. Privilege level -2 is reserved for special system functions and is used to implement TEEs that rely on hardware to set up access permissions. The TEEs that execute on this level share the processor with the OS. At Privilege level -3 the code is executed on a separate processor. This is the level at which TPM operate, as they, per definition, have their own coprocessors. Figure 2.4 shows the new software privilege levels and what they are commonly used for. Additionally, it shows which prominent TEE execute at which level, including Intel SGX and TrustZone which will be further discussed in this thesis. Bastion [9] and AEGIS [10] are two other TEEs depicted in the picture. The reader can further examine in their respective papers if interested.

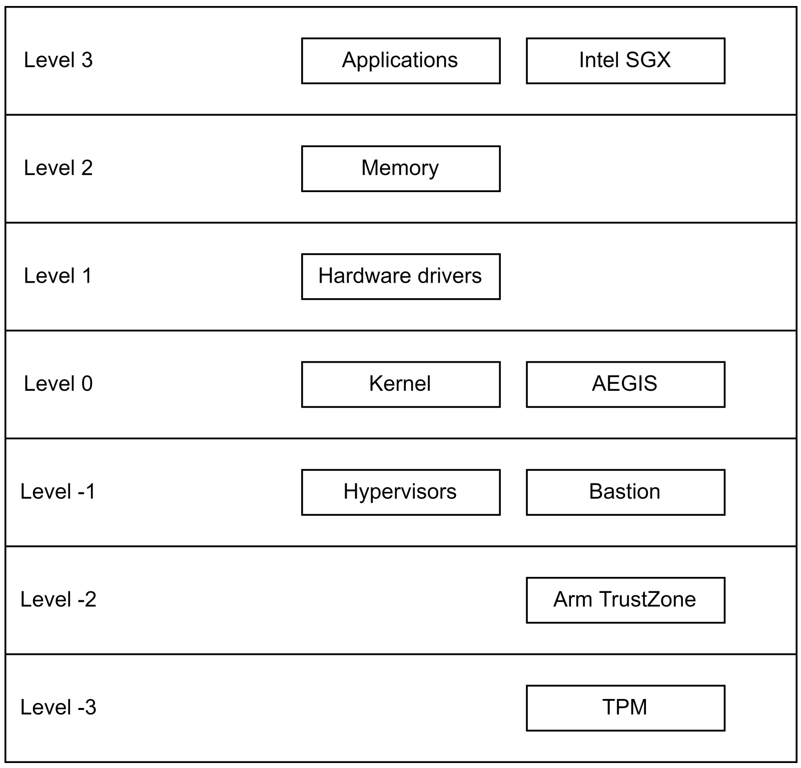

Figure 2.4: alternative software privilege levels

The secure mode is generally set up when the computer is started. When the computer boots its processor, it enters secure mode and setups internal data structures and protection for secure memory regions and peripherals. After this is done, the computer finishes booting and the secure mode can be entered. Entering is normally done by calling specific instructions.

2.2.3 Secure Storage

Secure storage is another core component of TEE. It is the memory where the confidentiality and integrity of the stored information are guaranteed. Without having a place to store important data such as cryptographic keys or other primitives, isolated execution can not provide any security guarantees, as attackers could just run another execution of the code with the same inputs. For this reason, secure storage is a major concern for any TEE technology. Secure storage can be achieved in many different ways. One common solution is to store the information in memory, but encrypt it using private keys that are stored within the respective TEE and not accessible to the outside world. Assuming the underlying encryption scheme is secure attackers are able to get access to the encrypted data but are unable to decrypt it and acquire the details. While this solution protects most of the relevant information, some details are still available to the attackers, such as when it was written and subsequently when the TEE was executing, and the approximate amount of information. Other approaches include but are not limited to using level -2 system functions to define memory ranges which are only able to be accessed during the secure mode. ARM TrustZone uses this approach.

2.2.4 Remote Attestation

Another important component of TEE is remote attestation, which is the ability of a TEE to prove to third parties that it has not been compromised and its result can be trusted. This is important as third parties rely on the fact that the surrounding potentially malicious OS is unable to tamper with the execution happening inside the TEE to ensure important information is not leaked. A way to implement this is using one time writable memory to store secrets at the time of manufacturing and an attestation service provided by the manufacturer. By creating a hash of the code to be executed and encrypting it using the stored secrets it is possible to verify the execution using the attestation service provided by the manufacturer. As this solution is rather complex a more detailed explanation is given in Section 3.2.3 where it is implemented by Intel SGX.

2.2.5 Secure Provisioning

Secure provisioning goal is to be able to send information to the TEE while protecting its secrecy and integrity. This is important as TEE are part of a larger untrusted system, which may be able to read or even intercept any data sent through it. To prevent this, there are many approaches. One of them is to use remote attestation to confirm that a public encryption key belongs to the environment in question and use it to encrypt the data which can then not be read or changed by the attacker. Another solution used by Intel SGX is the Diffie-Hellman key exchange. It uses the public and private keys of the participating users to generate a shared secret, which can then be used to perform symmetric encryption on any data transferred between them. This approach will be further described later on.

2.2.6 Trusted Path

The last feature discussed in the paper by Vasudevan is the trusted path, the ability to set up an isolated channel for communication between the TEE enabled processor and a peripheral from which it can communicate with the outside world. This is important for third parties, as this allows them to be certain that they are interacting with the environment and not a spoofed one. While ARM TrustZone does not inherently support this feature, one of its extensions does. It is called peripheral isolation and partitions peripherals into secure and non-secure ones, with the secure peripherals being invisible to applications during normal execution.

2.2.7 Common Trusted Execution Environments

This section goes over some of the common TEEs and discusses their core aspects.
Intel SGX is the first technology to be discussed. It is commonly found in larger computers like desktops that include Intel made processors. In contrast to common TEE implementations which execute at more privileged levels to protect itself from malicious system software, Intel SGX executes its applications at Ring 3. It achieves isolated execution by having a special processor mode called enclave mode that enables the execution of important applications at Ring 3 while being isolated from the system software that executes at Ring 0. Furthermore, it uses a special memory region that is inaccessible for untrusted parties to protect itself against DRAM attacks. Additionally, it possesses an inbuild remote attestation protocol based on Enhanced Privacy ID (EPID), which will be discussed in detail later on. It should be mentioned that while Intel SGX security aspects are sufficient for most applications, it still has security flaws, mainly side channel attacks. This issue is addressed by Sanctum, another TEE that similarly executes at Ring 3[11]. It also possesses an enclave mode that allows isolated execution of applications at ring 3, but in contrast to Intel SGX, Sanctum does not have encrypted memory, making it vulnerable to attacks targeting the DRAM. Sanctum improves upon Intel SGX security issues concerning side channel attacks but is currently not commercially available.
AEGIS is one of the first TEE, proposed in 2003 by Suh [10], 12 years prior to Intel SGXs introduction. It executes at ring 0 by separating the OS into a trustworthy part, called the security kernel, and an untrusted section. This partition can happen on a software level, but it is also possible to implement it on a hardware level. Important applications are then executed in a tamper-evident environment or a tamper-resistant environment, AEGIS's version of a secure mode, through which it achieves isolated execution. These environments can either detect memory tampering attempts by untrusted software or can prevent them in the tamper-resistant environment and can therefore enable secure storage. It also possesses a remote attestation scheme, called certified execution, in which a tamper-evident environment produces a certificate confirming that a calculation was executed by the environment which can then be verified by the third party.
An example for a TEE operating at ring -1 is Bastion, which was introduced by Champagne and Lee in 2010 [9]. It is based on a trusted hypervisor that is able to allocate resources to important applications and execute them in isolation from surrounding software, achieving isolated execution. Additionally, secure storage is guaranteed by encrypting memory stored on unsafe devices, preventing attacks that target the secrecy of the data. It does not possess any remote attestation scheme.
ARM TrustZone is a prominent example of a TEE executing at ring level -2. It can safely execute security sensitive code in its secure environment which is segregated from the surrounding code. Even untrusted hypervisors are unable to negatively impact it. By partitioning memory regions or devices into secure and nonsecure ARM TrustZone is able to provide secure storage for any cryptographic primitives and other important data. Similar to Bastion ARM TrustZone does not feature a remote attestation scheme, but solutions to this can be implemented which will be discussed later. Another technology that falls into the ring -2 TEE is the System Management Mode developed by Intel for x86 platforms [12]. While not inherently a TEE it is capable of providing an isolated execution environment for small TEEs. However, it does not feature secure storage or remote attestation because of its size.
The most prominent TEE that executes at Ring -3 is the TPM [13]. It is able to safely store cryptographic primitives, such as software measurements. However, it is unable to provide an execution environment that can be used to safely execute applications. Nevertheless, the cryptographic primitives stored inside the TPM can be used to instantiate trusted hypervisors, ensuring a certain level

of trust for the platform the TPM is part of. The TPM can also be used for remote attestation as it can store keys used for it in its secure storage. The Intel Management Engine [14] is another example for a TEE technology at ring -3, unlike the TPM it is able to execute code but does not feature secure storage or remote attestation. Table 2.2 gives a overview over the presented TEE and their features.

Table 2.2: common trusted execution environments and their features

TEE	Ring	Isolated Execution	Secure Storage	Remote Attestation
Intel SGX	3	yes	yes	yes
Sanctum	3	yes	yes	yes
AEGIS	0	yes	yes	yes
Bastion	-1	yes	yes	no
ARM TrustZone	-2	yes	no	no
TPM	-3	no	yes	yes
Intel ME	-3	yes	no	no

Of the presented TEE technologies, Intel SGX and ARM TrustZone are the most important, and therefore the target of comparison in this thesis. The reason is that while there are many technologies that enable TEE they are not popular. Some technologies such as Sanctum are not available to the general public and others are only used for research purposes. The relevant TEE for the general consumer can be limited down to a few environments, with both ARM TrustZone and Intel SGX being part of them. Intel SGX is included in most of the modern Intel processors available on the market. Because Intel holds a large market share compared to other computer processors it is of greater interest [15]. For this reason, it is one of the environments to be compared in this thesis. ARM TrustZone, the other TEE compared in this thesis, was chosen for similar reasons. In contrast to Intel SGX, ARM TrustZone is more commonly found on smartphone processors and IoT devices. It holds more than about 90 percent of the market share in both sectors and is continuously increasing its share in sectors such as cloud processors and car assistance [16]. This thesis presents some of the reasons for their success and why they are used by so many consumers.

3 Overview of Intel SGX and ARM TrustZone

This chapter goes over the two hardware TEE technologies that are the topic of this thesis. The first section goes over ARM TrustZone, a hardware based TEE enabler that uses a secure area in the main processor to enable important features. The second section introduces Intel SGX, a hardware TEE technology that uses a special processor mode, privileged supervisors, and cryptographic encryption to provide isolated execution and secure storage. Additionally, it also possesses a remote attestation scheme that will be described here.

3.1 ARM TrustZone

ARM TrustZone is a hardware TEE enabling technology developed by ARM since 2004 and distributed in most ARM designed processors. These processors include but are not limited to the Snapdragon series and the Apple Silicon series, both of which see widespread use in smartphones developed based on Android and iOS respectively. This technology is based on two worlds in which the processor can execute, referred to as normal world and secure world which will be further described in Section 3.1.1. Another key component is the ability of TEE enabled processors to partition the memory, no matter which kind, into those accessible during the secure world and those that can be used while in the normal world, which will be extended in Section 3.1.2. Finally, in Section 3.1.3 this thesis describes features that are not inherently included in TrustZone processors but can be included by adding additional components or software. The information in this section were provided by the papers TrustZone Explained by Ngabonziza [17] and Demystifying ARM TrustZone by Pinto and Santos [18]. Interested readers may go over them for more information.

3.1.1 Normal World and Secure World

The core feature of ARM TrustZone is its processor's ability to switch between the secure world and the normal world. These worlds are managed by their own OS and provide the same functionality, e.g., it is still possible to execute multiple applications in the secure world. The main difference between them is the security guarantees provided. The processor can only execute in one of the two worlds, determined by the value of a special bit, referred to as the Non-Secure (NS) bit. The value of this bit is known by the whole system through the Secure Configuration Register (SCR) from which memory and peripherals can read it, but cannot be changed by them.
Switching between both modes depends on the specific ARM architecture. In the Cortex-A architecture, which is the architecture used for performance-intensive systems like smartphones [19], the secure world is accessed through a special processor mode called secure monitor, which connects both worlds and is the sole point of entry to the secure world. While in the normal world the

instruction Secure Monitor Call (SMC) allows the processor to switch from normal world to secure world through the secure monitor. Certain exceptions or interrupts which pass through the secure monitor are also handled in the secure world. This setup is shown in Figure 3.1.

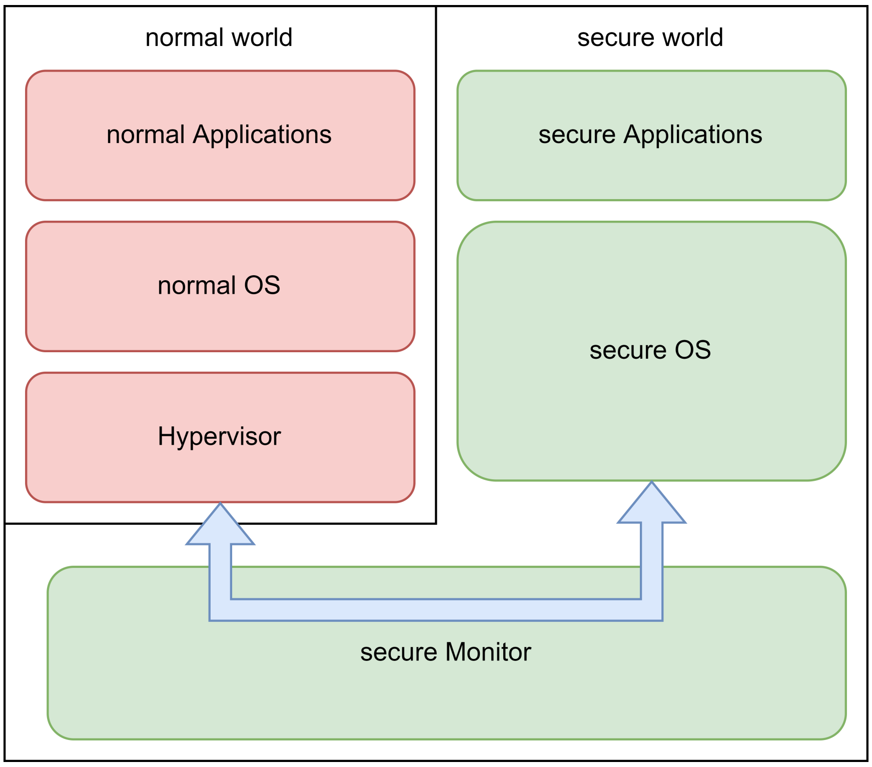

Figure 3.1: Cortex-A worlds

Cortex-M architecture is commonly used in embedded applications [19], its world switching is achieved in a different fashion. As Cortex-M has an emphasis on being faster and less resource intensive it skips the monitor mode to enable faster world transitions. Instead, this architecture supports three new instructions that allow for world switches. The first one is Secure Gateway (SG) which is used to enter the secure world from the normal world. The second is called branch with exchange to non-secure state (BXNS). It is called when a program wants to return to the normal world from the secure world. The last one is used when calling functions in the normal world from the secure world. It is referred to as branch and link with exchange to non-secure state (BLXNS). This new architecture is displayed in Figure 3.2.

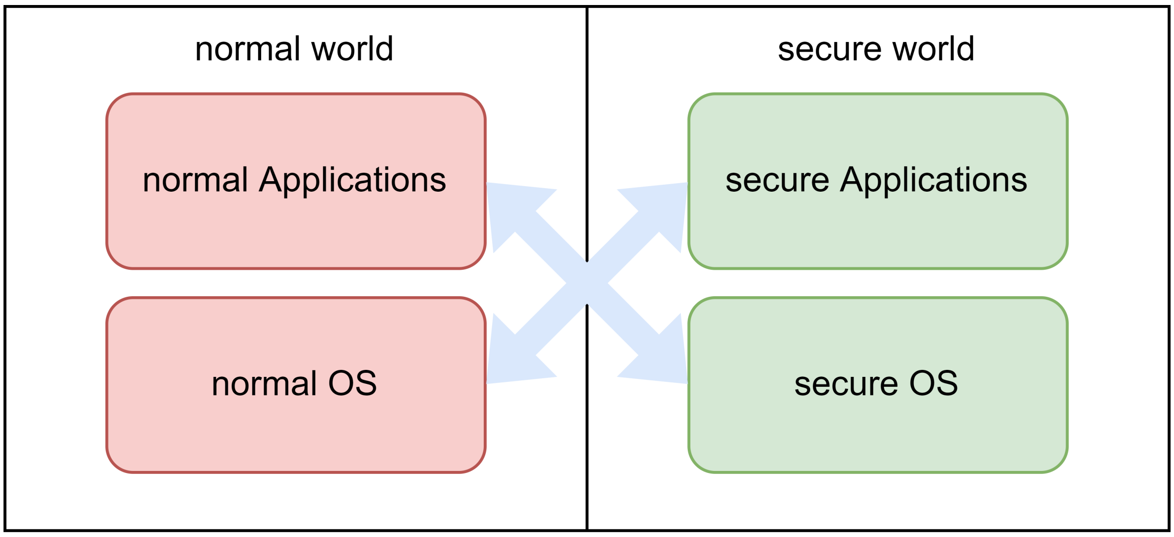

Figure 3.2: Cortex-M worlds

These different worlds are separated physically by having specific registers only be visible in the secure world and inaccessible in the normal world. Other hardware such as memory or peripherals can be set to be only accessible in specific worlds by the producer of a system. These are then commonly enforced by special access control hardware that are aware of the existence of the TrustZone during normal mode and prevent software from accessing hardware they should not be able to in the normal world. An example configuration can be seen in Figure 3.3. This example shows a configuration in which access to certain hardware, in this case, part of the cache, the DRAM, and all peripherals, is only possible while the processor is in the secure world. It should be mentioned that the use of access control hardware is optional and there exist processors based on ARM architecture that do not make use of them, opening potential attacks which are described in Chapter 4.

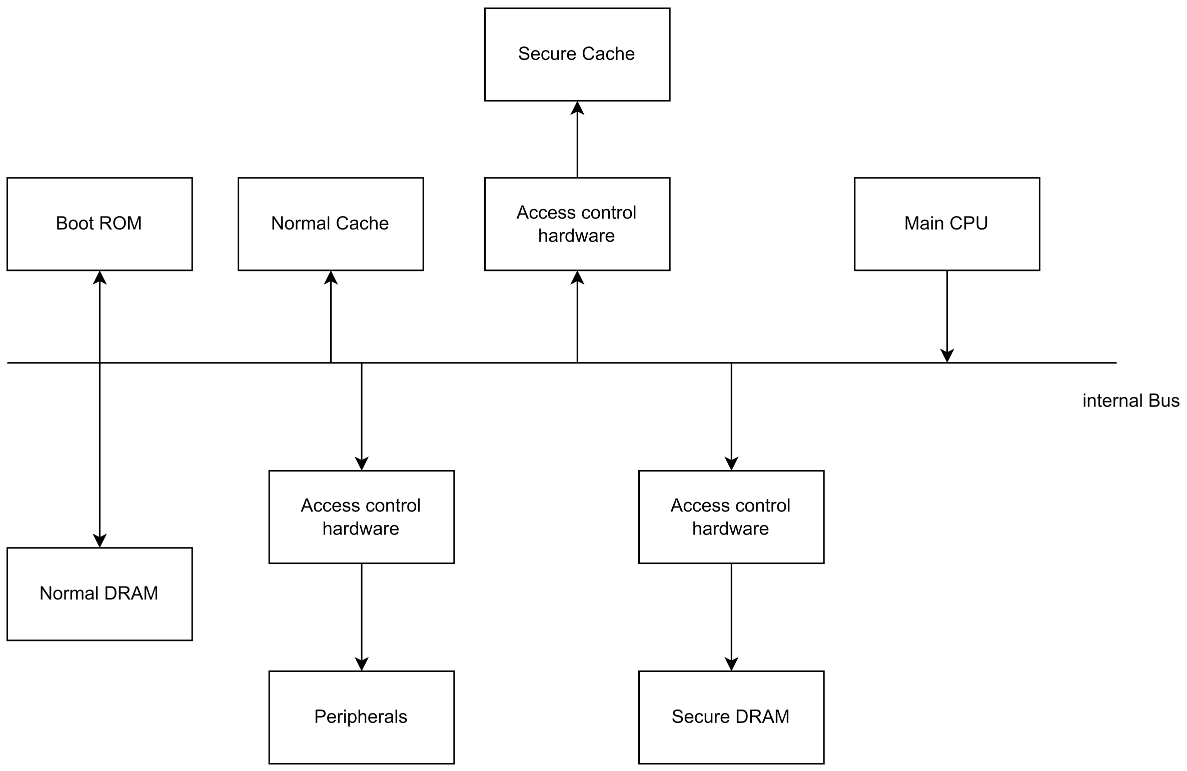

Figure 3.3: example access configuration

3.1.2 Memory Partitioning

To achieve secure storage ARM TrustZone partitions its memory into secure areas and non-secure areas, with the nonsecure area being used for memory and devices accessible from the normal world and the secure area reserved for those only usable in the secure world. This is achieved through the use of the TrustZone Address Space Controller (TZASC) and the TrustZone Memory Adapter (TZMA). The TZASC is responsible for classifying the DRAM into secure and non-secure zones. It is controlled by the secure world and can be used by software running in it. The TZMA is able to divide the cache or other on-chip memory into secure and non-secure areas, which have to be in multiples of four KB. The Cortex-M architecture further divides the secure area of the memory into secure memory and non-secure callable. This new non-secure callable memory area holds special SG instructions that allow for the transition between secure world and normal world. These additional memory sections were introduced to prevent attackers from using binary data with the same opcode as the SG instruction to enter the secure world. These additional gateways and controllers are optional in ARM TrustZone architectures. Developers can decide to forego them in favor of smaller and less power hungry processors. However, more recent processors that support TrustZone tend to include these to provide secure storage for a TEE.

3.1.3 Other Components

One core component of TEE that is not inherently supported by ARM TrustZone is remote attestation. However, support for it can be included by the manufacture of the processor. One possible solution is presented by Wang, Zhuang, and Yan called TZ-MRAS [20]. This solution requires two new components, a report module and a measurement module, in addition to an existing secure storage located on a TPM. The scheme starts by having a verifier generate a nonce and sending it to the prover, who uses the nonce to generate a report in its report module. The report includes information about the current platform configuration in hash form and the attestation path of the TEE measured by the measurement module and signed by the report module using the TPM. For this signature, the report module uses an identity verification key called the AIK which is stored in the secure storage. This report is then sent back to the verifier who can then confirm the identity of the prover by checking the AIK signature and confirm the integrity of the TEE by checking the measurement of the TEE with the expected value. It should be noted that the generation of signature and the secure storage are located on a TPM, which may not be available on all platforms, but can be implemented using ARM TrustZone on a software basis as presented by Raj called fTPM [21].
Other important components that are commonly part of ARM TrustZone based architectures are the Generic Interrupt Controller (GIC) and the TrustZone Protection Controller (TZPC). The TZPC allows for the separation of devices into those accessible in the secure world and those accessible in the normal world, providing a trusted path from the TEE to the devices. It achieves this by setting three 2-bit registers which allow for it to control up to 8 signals. The GIC is responsible for controlling both secure and non-secure interrupts. It prevents non-secure interrupts from maliciously affecting the secure world and protects itself from Denial of Service (Dos) attacks by prioritizing certain interrupts and therefore ignoring less prioritized non-secure interrupts which may be used against it. Furthermore, ARM TrustZone can support secure boot or trusted boot by authenticating each software image executed during the startup sequence of a computer and preventing malicious code from being run.

3.2 Intel SGX

This section goes over Intel SGX and gives a high level overview of it. If the reader is interested in a more detailed description, this thesis points to the Intel Software Developer Manual[12]. A shorter summary can also be found in the paper Intel SGX explained by Costan [22]. Both sources were used during the writing of this thesis.
Intel SGX, short for Intel Software Guard Extension, was introduced by Intel in 2015 with its sixth generation of Intel Core microprocessors. It is included in many modern laptops and desktop computers which use Intel processors, ranging from vendors such as Lenovo to Dell.
Intel SGX revolves around the enclave, a separated and encrypted region for code and data. Code can be executed in an enclave, isolated from potentially malicious software. It provides confidentiality and integrity. This is achieved by many different components and protocols. One of those is a special memory region, called the Processor Reserved Memory (PRM), which provides secure storage that can not be accessed by other software even at the kernel level. This memory is further described in Section 3.2.1. To execute code in a secure mode the processor loads code from outside into the PRM. After this is done the processor uses special instructions to enter secure mode, called enclave mode, and execute the loaded code. This is further specified in Section 3.2.2. The remote attestation scheme Intel SGX uses is described in Section 3.2.3.

3.2.1 Processor Reserved Memory

The Intel SGX's form of secure storage is called PRM. It is part of the DRAM and protected from access by non-enclave mode software by dedicated memory access controllers. This special memory is further subdivided. One partition is the Enclave Page Cache (EPC). This cache is further divided into 4 KB pages which store the code of the specific enclave they belong to, enabling the running of multiple enclaves in the system. The EPC is managed by the OS and as it is a subset of the PRM. Therefore, it can not be accessed by non-enclave software. As the OS is not trusted in TEE Intel SGX possesses a feature to supervise the OS interactions with the EPC, called the Enclave Page Cache Map (EPCM). The EPCM is an array with one entry for each page stored in the EPC, which stores information about the ownership, the virtual address, and the structure of the respective page. The security checks can then use the information to prevent malicious attack attempts by the OS, for example allocating the same page twice. Other details about the EPCM are not public [22], therefore the developer can mostly ignore it. Another important part of the EPC is the SGX Enclave Control Structure (SECS), which is stored in a special page part of the EPC. The SECS contains the metadata of their respective enclave and are therefore very important in defining its identity. These include attributes that influence the execution environments of the enclaves, part of which is the debug flag which enables the use of SGX debugging features and if enabled leads to the loss of all SGX security features. Because of this, the page that contains them possesses additional security measures which prevent them from being mapped into address space, making them only accessible to the SGX implementation. The structure of the memory is pictured in Figure 3.4.

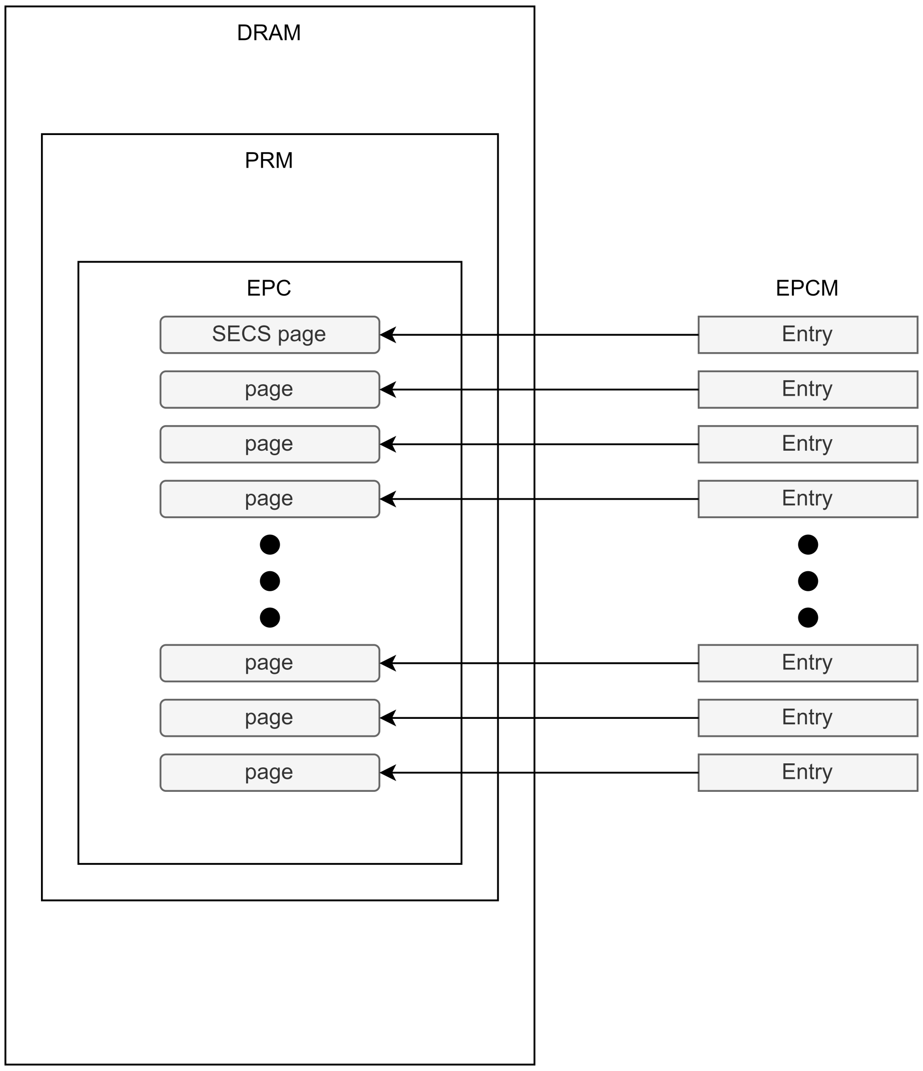

Figure 3.4: SGX memory layout

An enclave accesses the EPC by dedicating a part of its virtual memory to the Enclave Linear Address Range (ELRANGE), which contains the addresses which are mapped to the EPC. Other addresses in the virtual memory are mapped to memory outside the EPC. A core feature of the memory layout is that the processor checks that if an address translation results in a physical address of a page stored in the EPC the virtual address matches the one stored in the EPCM. This can prevent certain address translation attacks. Another security measure taken in this regard is that pages possess access permissions which are set at the allocation of the respective page and defined by the author of the enclave. This defines addresses that are allowed to read, write and execute enclave code. This information is stored in the EPCM. One more component in regards to the memory layout of Intel SGX is the State Save Area (SSA). It is a special memory region used to store the enclave codes execution context when an interrupt or exception occurs which necessitates the processor to exit enclave mode and enter normal mode. If the processor wants to continue executing the previous code it can then load the previous state from the SSA and resume the execution. Another feature important for security reasons and performance is the possibility to evict pages from the PRM to non-PRM memory. This is supported as most modern computers support the over commitment of memory by loading memory when required and evicting them when they are not used anymore, as described in Section 2.1.3. Intel SGX enables this by adding certain measures to guarantee the integrity and confidentiality of the evicted pages. To achieve this Intel SGX uses symmetric key encryption with nonces which are stored in the Version Arrays, which are special pages stored in the EPC. By having the memory stored outside the PRM be encrypted, SGX protects it from malicious reading attempts in unsafe memory.

3.2.2 Enclave Life Cycle

Before the enclave can be used, it first has to be created. This starts with calling the instruction ECREATE, which creates a new enclave and stores its metadata in a new SECS with its INIT value set to false. To add new pages to the new Enclave EADD is used. This results in the OS loading a new page into the EPC and setting its VALID value to 1, indicating that it is allocated. It should be mentioned that attempting to add additional pages after this stage will result in an error. Additionally, the OS will calculate a new measurement for the enclave by using the instruction EEXTEND 16 times which will be used for attestation purposes described later. When the enclave pages are all allocated, the processor can start executing them. This happens by calling EINIT and by providing the instruction with a token that is received from the Launch Enclave. The enclave's INIT value is then set to true. Applications executing at Level 3 can now execute the enclave code. The Launch Enclave is a special enclave provided by Intel, which also goes through the same step as other enclaves to initialize but skips the token step in EINIT. It is solely used to provide launch tokens to other enclaves based on a white list of approved enclaves, which is partially managed by Intel. This means that any company that wants to use an enclave needs to be approved by Intel. Applications can execute the enclave if they have the respective EPC pages in their virtual memory. They can then call EENTER to enter enclave mode and execute enclave code, and exit it when they are done using EEXIT. If an exception or interrupt occurs during execution in enclave mode, the processor can exit it using AEX which saves the current execution context in an SSA. After the interrupt or exception is handled the processor can use ERESUME to resume its execution using the execution context stored in the SSA. It should be mentioned that an enclave can have more than one SSAs to store multiple execution contexts in the scenario of multiple interrupts happening while executing the same enclave. After the execution is done, the pages associated with the enclave are evicted by setting the VALID value in the respective EPCM entry to zero and the TLB is flushed to prevent memory attacks. Figure 3.5 illustrates this process with an enclave that is in possession of a single SSA.

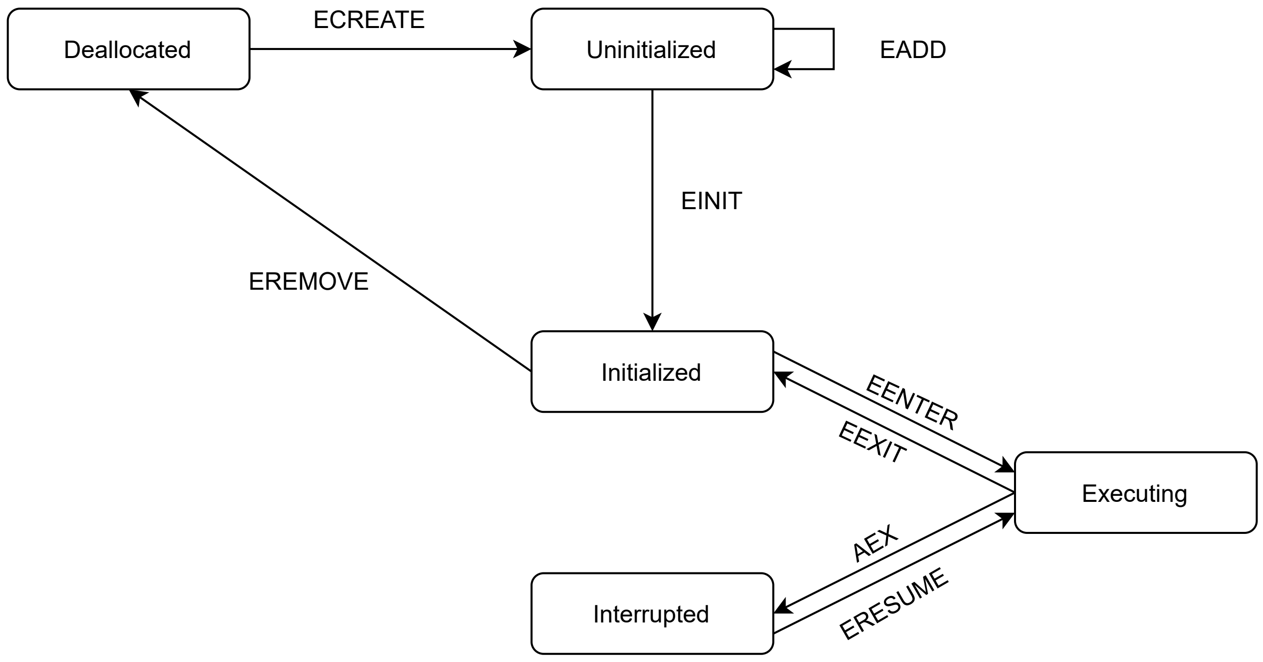

Figure 3.5: life cycle of an enclave

3.2.3 Local and Remote Attestation

Intel SGX supports two different attestation schemes. The first scheme, local attestation, is used to establish a channel between two different enclaves on the same device that guarantees confidentiality, integrity, and replay protection. The second scheme, remote attestation, is used to prove the integrity and intactness of the hardware in question, in this case, the enclave, to a third party that is not situated on the same device and gain his trust. This subsection will first go over local attestation and then describe remote attestation.
Local attestation is used if two different enclaves on the same device want to work together. For this, they require a secure channel through which they can communicate. For this scheme, Intel SGX uses a security protocol called Diffie-Hellman Key Exchange, which is used to share a symmetric key through an untrusted channel.
Diffie-Hellman can be summarized as follows as an analogy with colours, but in practice, it is used with very large numbers and mathematic operations. The example shown here is depicted in Figure 3.6. Imagine two parties, called Alice and Bob, attempting to establish a communication channel using a symmetric key. Both parties begin the Diffie-Hellman Key Exchange by sharing a common paint, in this example blue, through an untrusted communication channel. The common paint is considered known to potential attackers. Both parties also possess a secret colour that is only known to themselves and they proceed to mix their secret colour with the common colour blue. In this example, Alice has the secret colour yellow, mixed with blue it results in green colour. Bob has the secret colour red which mixed with blue makes magenta. These mixed colours are then shared again between both parties through the untrusted channel, resulting in the attacker now knowing the common colour blue and the mixes of both parties green and magenta. Alice and Bob can now mix the colour they received from the other party with their own secret colour to make

brown, which is then used as the symmetric key for both parties as green mixed with red results in brown and magenta mixed with yellow results in brown. As the attacker is only aware of blue, magenta, and green. He is unable to use them to create the colour brown and determine the secret key. This protocol works under the assumption that it is very difficult for the attacker to derive the secret colour of either party while only knowing the mix of the colour and the common component of it.

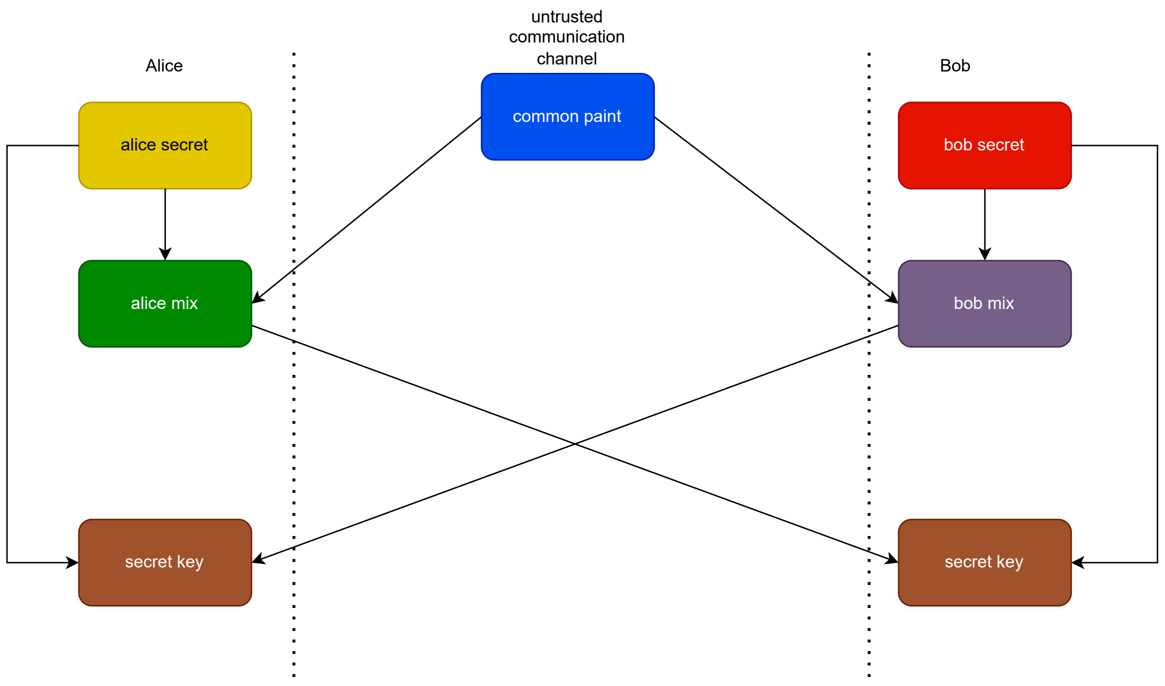

Figure 3.6: diffie hellman example using colours

Intel SGX Local Attestation scheme begins with one enclave sending its identity, the MRENCLAVE value, to another enclave it presumes to be on the same device. The sender is from now on referred to as the verifier as he wants to verify that the receiver is on the same platform, who in turn is from now on referred to as the claimer. The claimer now uses the MRENCLAVE value he received from the verifier to produce a report which it sends back to the verifier. This report can be verified by the REPORT KEY which is stored on the device and therefore accessible to all enclaves that are on it, it also contains Diffie-Hellman Key Exchange data which can be used to establish a secure communication channel. After the verifier gets the report, he can therefore verify it using the REPORT KEY. After the verification, he too creates a report for the claimer so that the claimer can verify that the verifier is also on the same platform. Both parties can then create a secure channel using the Diffie-Hellman Data contained in both reports. Figure 3.7 shows the local attestation scheme in Intel SGX.

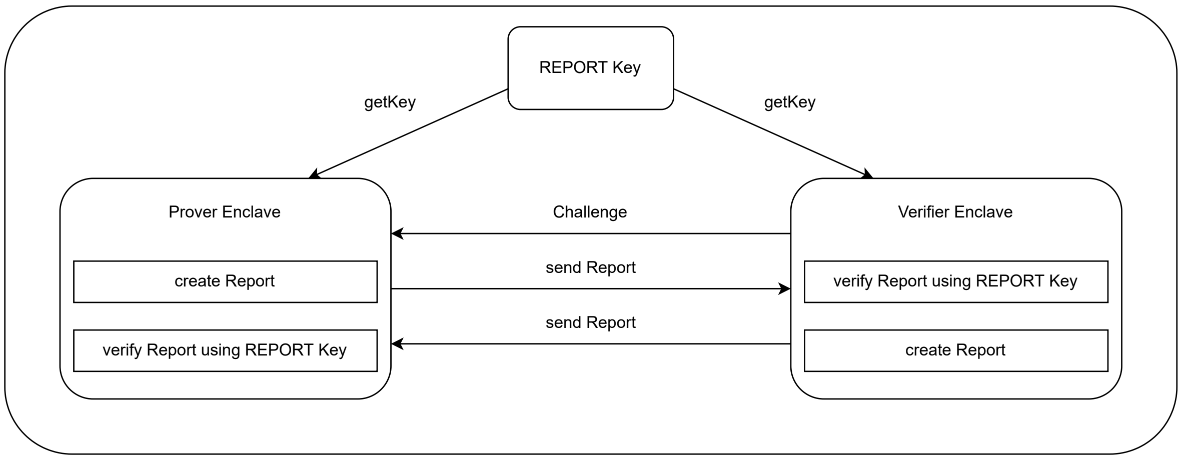

Figure 3.7: local attestation in Intel SGX

To achieve remote attestation, Intel has included different components into their processors, including one time writable memory and special enclaves. One important component of remote attestation is the enclave measurement, which was already mentioned previously. The enclave measurement is used to identify the software executing inside, and the third party compares the measurement with an expected value to verify that the code executed inside the enclave is the one the third party wants. The measurement consists of many different parts and is computed using the secure hash functions SHA-2 [23] on the inputs of the ECREATE, EADD, and EEXTEND instructions in an order specified by the author of the enclave. After EINIT is called, the measurement cannot be changed anymore. Using this, the third party can be sure that the code inside the enclave is correct and has not been tampered with. Another important aspect is that the code is running inside an intact Intel SGX enclave. To attest this SGX uses an Intel Attestation Service run by Intel, which is used to verify the integrity of an SGX enclave. This is done by verifying the quote provided by the quoting enclave which contains the local attestation report of the proving enclave. Other important components used in the remote attestation process are inbuilt secrets and the provisioning enclave. The secrets that are inbuilt at the time of manufacturing are the seal secret and the provisioning secret, with the provisioning secret being generated by Intel and the Seal Secret being generated inside the processor and consequently unknown to Intel. The provisioning enclave can then use a provisioning key derived from the provisioning secret to authenticate itself to an Intel Provisioning service, proving that it can be trusted. The Provisioning service then sends the enclave an attestation key. The remote attestation process starts by having the challenger send out a request to the third party which includes its enhanced private ID. The ID is used for the key exchange based on the SIGMA protocol [24]. The third party replies with a challenge for the enclave that creates a report. The report is forwarded to the quoting enclave, who then uses local attestation to verify the enclave attempting to attest. The quoting enclave can then use the attestation key from the provisioning process to create a quote including the measurement and other proofs, for example, that the key has not been revoked. The quote is encrypted using the key of the Intel Attestation Service. The quote is then sent to the attesting enclave who forwards it to the third party. As the quote is encrypted, the third party has to forward the quote to the Intel Attestation Service that checks that the platform has not been revoked before reporting the quote to the third party who can check the measurement. This process ensures that the enclave is properly running using the latest SGX security measures and executing a specific piece of code. Figure 3.8 displays the steps in this process.

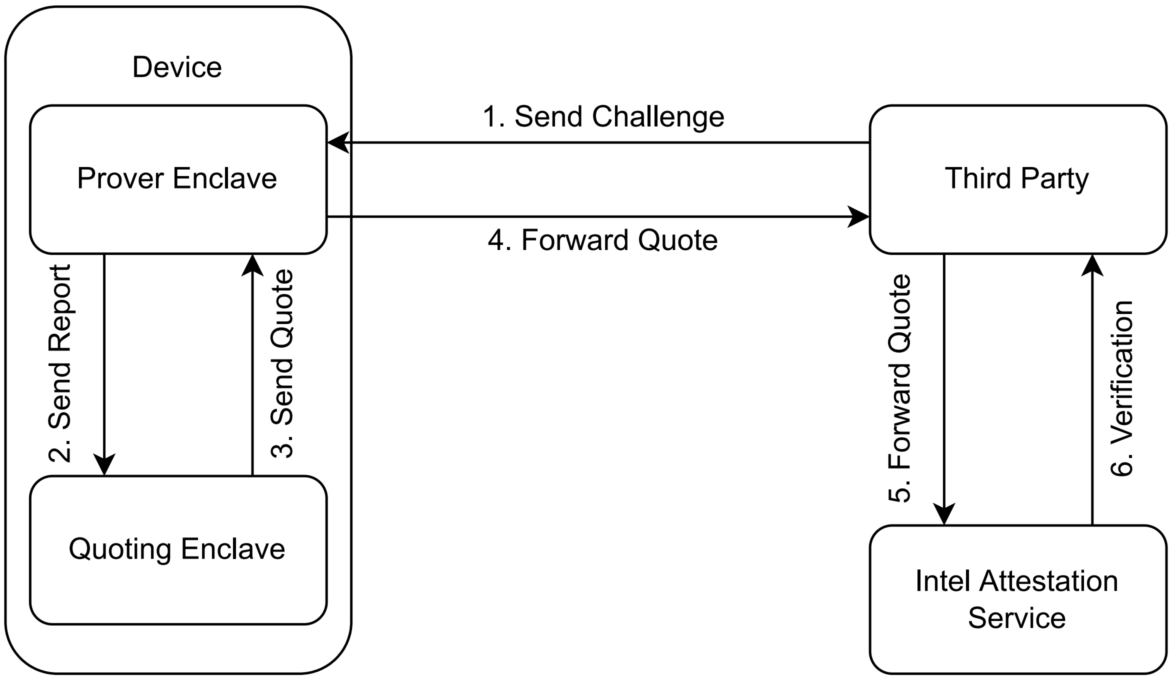

Figure 3.8: remote attestation in Intel SGX

3.3 Differences between Intel SGX and ARM TrustZone

The goal that Intel SGX and ARM TrustZone want to achieve is the same. Both technologies want to enable the execution of security sensitive applications on devices without having to rely on the integrity of the device in question. To achieve isolated execution, both environments use a special processor mode, referred to as enclave mode in Intel SGX, and secure world in ARM TrustZone, which allows for the execution of applications in isolation of surrounding system software. Both TEE also enable the execution of multiple applications at the same time, Intel SGX enabling the creation of multiple enclaves. ARM TrustZone can execute multiple applications while in the secure world. Where both environments differ in terms of isolated execution are in regards to their execution levels. Intel SGXs enclaves execute at ring level 3, with more privileged components ensuring the integrity of the execution. In contrast ARM TrustZone executes at a higher privilege level than the system software, at ring level -2 in a mode that is inaccessible and invisible to the untrusted OS.
Secure Storage is also achieved in a different fashion. Intel SGX uses a special memory region, the PRM, to save important information and relies on encrypted memory in an unsafe memory location to achieve a persistent secure storage. This leaves Intel SGX vulnerable to passive memory attacks where the attacker only listens to memory accesses, as the memory is managed by untrusted system software. ARM TrustZone is able to solve this by partitioning memory and memory devices into secure and non-secure ones. Non-secure applications are unable to access the secure memory or even see it. Because TrustZone also uses a secure OS to manage memory passive attacks are not feasible as an attack vector.
Another large contrast is regarding Remote Attestation. While Intel SGX provides its own attestation

scheme using EPID which provides additional anonymity features and revocation features, ARM TrustZone does not. This is because ARM TrustZone is more commonly used as a TEE enabling technology and not as a TEE. Many TEE implementations, for example, Samsung Knox developed by Samsung, use ARM TrustZones secure world to implement more applications, ranging from core usability features like remote attestation to performance features like Digital Rights Management. Intel SGX on the other hand provides most of the features already desired by TEE. Software TEE that are based on SGX are less common but still exist.

4 Comparison when facing common security threats

This chapter compares Intel SGX and ARM TrustZone in terms of how they perform against common security threats. For this purpose, this thesis will first summarize the main security features of ARM TrustZone and Intel SGX. The second section goes over the main attacks that can be used against TEE. The third section presents common security threats that can be used against TEE. The final section contains the comparison.

4.1 Security Goal of TEE

The main goal of TEE is the ability to provide an environment in which applications can be executed safely and cryptographic primitives can be stored in a secure manner. For this reason, most TEE aim to provide an isolated execution environment, in which the integrity and confidentiality of the executed code can be guaranteed. Secure storage guarantees the integrity of the stored data such as keys used for encryption or signing purposes. Remote attestation proves the integrity of the environment to third parties. Therefore, the end goal of TEE technologies is to guarantee the integrity and confidentiality of code, data, and itself. For this reason, the attacks in this section only consider attacks threatening the isolated execution and secure storage. Attacks that threaten the trusted path or secure provisioning are therefore not considered.

4.2 Overview of security features of ARM TrustZone and Intel SGX

The core security feature of ARM TrustZone is the separation of the system into two different worlds: a normal world that is considered insecure and therefore untrusted; and a trusted secure world. When the processor is executing inside the secure world, untrusted applications are unable to interfere with it, at the same time, the processor can have access to devices that were previously inaccessible, including secure memory or I/O devices. The secure memory is protected by the TZASC and the TZMA. For this reason, memory access from the normal world is not possible. Additionally, it is managed by the secure OS running in the secure world, meaning a compromised normal OS is unable to interact with the secure storage. The TZPC is responsible for the I/O devices that are only accessible inside the secure world. It also enables a trusted path to the outside world. Another potential security feature that ARM TrustZone can have is the secure boot, which prevents malicious code from being run before the security mechanism is enforced by the system.
Intel SGX security guarantees revolve around the enclave, in which code can be executed isolated from untrusted system software. The first core security feature of Intel SGX is memory isolation. The PRM allows for the definition of an area of memory on the DRAM at boot time that is not

directly accessible for applications outside of those in enclave mode and the OS. While the system software is unable to directly access it, it still manages the PRM using special instructions. This makes SGX vulnerable to passive memory attacks. Another important security feature is the enclave measurement. It is used to establish the identity of the enclave, which can then be verified by a third party to check the integrity of the enclave. This enforces that the code inside the enclave is initialized in predetermined steps and executed properly. The third security feature is the SGX's ability to provide software attestation. This allows it to show third parties that it has been properly instantiated, prove its identity and that it can execute code isolated from surrounding software. The final security feature is the sealing of pages evicted from the PRM. By using symmetric encryption with nonces, these pages are inaccessible to malicious attacks even though they are stored in unprotected memory. The implementation differences of the core TEE features are summarized in Table 4.1

Table 4.1: Implementation of the core security features in Intel SGX and ARM TrustZone

Feature	**Intel SGX**	**ARM TrustZone**
isolated execution	applications execute in enclave mode	applications execute in the secure world
secure storage	important information stored inside the PRM, persistent storage through encryption in non-secure memory	secure memory devices with access management by the secure world aware TZASC and the TZMA
remote attestation	supported, using EPID	not supported but can be implemented using software

4.3 Common Security Threats

Even with the reduced attack surface, the TEE has compared to normal system software, there are still many possible attack vectors that can be used against it. While not all of them result in remote code execution, losing secrets stored in the TEE can still be devastating. Even knowing when and where a TEE is executing may result in potential danger for its integrity. This section describes some common attack patterns and gives a few examples of how they work. These range from physical attacks over privileged software attacks to address translation attacks. A more detailed summary of the presented attacks can be found in the paper by Costan [22].

4.3.1 Physical Attacks

Physical attacks generally involve the use of hardware or the exploiting of hardware. They can be categorized from noninvasive to invasive, depending on the degree of hardware manipulation. A simple denial of service attack that falls under this category is simply disconnecting the power supply of a computer or other electrical hardware and preventing it from being used. Other more advanced techniques include connecting USB drives through ports and starting a cold boot to gain access to a system's peripherals and steal encryption keys. Another physical attack is the bus tapping attack. By monitoring the bus on a computer motherboard, attackers are able to eavesdrop on the traffic, potentially even modifying it with malicious intent by injecting new commands or replaying old ones. Another approach to physical attacks is monitoring the power consumption of the processor and inferring the type of computation currently happening inside it. This type of attack is called the power analysis attack and can also be used against TEE, as they rarely alter their consumption habits to mask their current computation. The last class of physical attacks discussed in this thesis is chip imaging attacks. These attacks use advanced tools, such as ionbeam machines to enable semi-invasive attacks on chips. They can then control the behaviour of hardware chips and circumvent the protection mechanism. They do not damage the chip and are therefore considered semi-invasive. More invasive attacks such as reverse engineering and microprobing are not considered in this thesis. These types of attacks are generally considered costly, as most of them require access to the victim's device, which is very difficult for most attackers.

4.3.2 Privileged Software Attacks

The privileged software attack is one of the main concerns for all TEE. It assumes that attacks can happen from all privilege levels, even from the OS or other more privileged levels. These types of attacks include gaining access to the System Management Mode (SMM), which is the most privileged software mode on the computer and grants access to all software contained on it. While SMM was initially only accessible through the hardware, modern systems enable access to it through software. Some attacks exploit this vulnerability to gain access to the SMM and execute code there, which compromises the whole computer. Wojtczuk describes one such attack on Intel computers where the attacker provides the SMM with code to execute through a poisoned cache [25].

4.3.3 Software Attacks on Peripherals

Another form of attack is to use peripheral devices or their interfaces to gain access to memory or enable other attacks. In contrast to physical attacks, these attacks do not require hardware or physical attacks on the victim's computer and are therefore considered cheaper than them. An example of such an attack is the rowhammer attack [26]. The attack abuses a circuit level failure by repeatedly changing the content of a memory cell, which causes adjacent cells to change their value. The reason for this is that frequent activations cause adjacent rows to lose their charge more quickly than normal. This is depicted in Figure 4.1. By repeatedly changing the values of specific memory addresses the attacker could then alter data structures that were responsible for security decisions and then gain kernel level privileges. Another form of these kinds of attacks is the abuse of certain features for malicious purposes. An example is using the performance and temperature monitoring features of modern processors to gain information about the activity and current calculations of the processor. A more malicious approach is to use system software to alter the firmware of the computer and use it for a cold boot attack and generate massive security problems in the process.

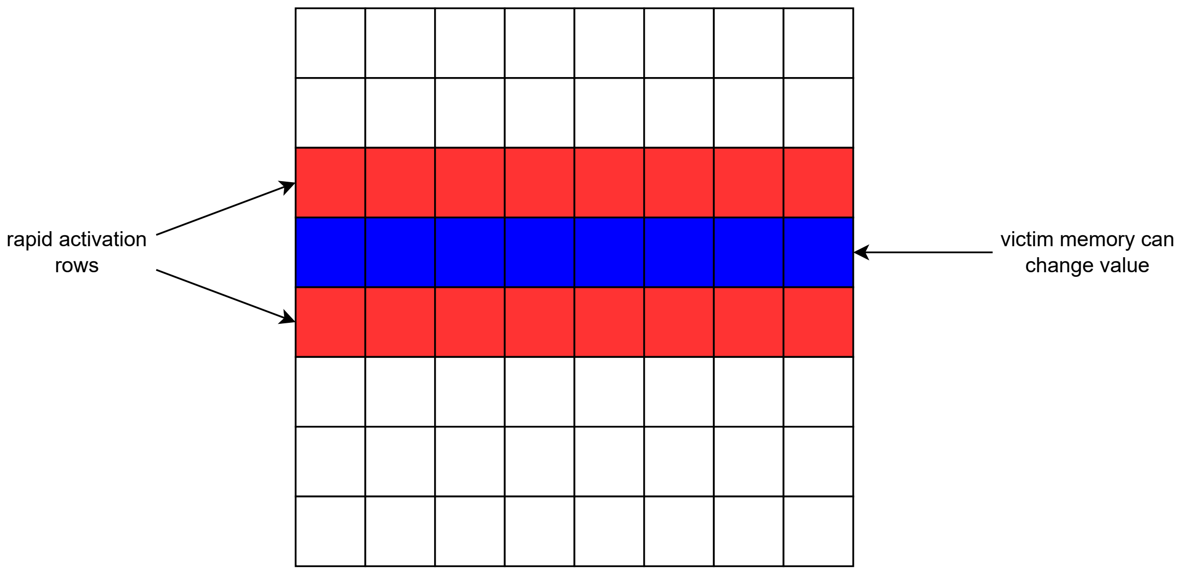

Figure 4.1: Rowhammer attack

4.3.4 Address Translation Attacks

Another major concern for TEE are address translation attacks, especially for Intel SGX, as its address translation process is managed by the untrusted system software in contrast to ARM TrustZone where it is managed by a trusted OS. As system memory is commonly managed by the OS, which is not trusted by TEE. It opens up attacks patterns executed by the OS. These can happen passively, by simply having the OS monitor access patterns of the software in the isolated environment and inferring information from it. To active ones that include the OS rearranging the memory in such a fashion that undesirable actions are taken. The OS can do this by switching the content of two memory addresses that contain different sets of instructions, with one set being the intended instructions and the other containing malicious instructions. The application that is attacked would then execute the malicious instructions, which may disclose secret information. Another point of weakness is when memory is evicted from the protected memory to the unprotected one. When the memory is reallocated into the protected memory the OS can simply switch the memory to facilitate defects and have the application execute the wrong set of instructions. TEEs have to protect from these attacks by not only tracking the correct virtual address for each physical memory, but they also have to bind each piece of memory evicted to the correct virtual address. However, even this may be insufficient, as address translation results are stored in the TLB and managed by the system software, which is mentioned in Section 2.1.3. This enables another more subtle attack: The malicious OS does not flush the TLB entries when an associated piece of memory is evicted. When the piece of memory is reallocated the OS can swap its content with malicious instructions. This is not detected as the TEE believes that the mapping stored inside the TLB is correct. For this reason, TEEs have to ensure that the TLB is flushed after the memory is evicted.

4.3.5 Cache Timing Attacks

The fifth form of attack described in this thesis are cache timing attacks. These attacks can be performed at ring 3. They abuse the fact that accessing memory that is already loaded is faster than accessing memory that is currently deallocated. By measuring the time difference using the system instructions available at ring 3, the attacker can determine whether its current access attempt points to allocated or unallocated memory. By filling up the cache with the attacker's memory and afterward allowing the process in a TEE to execute, the attacker can then monitor the access pattern of the software. The attacker does this by making repeated cache access attempts and determining whether they are hits or misses. In this way, it can determine which cache lines are currently being used by the victim, or even steal important secrets of the victim. This approach is called prime and probe and is illustrated in Figure 4.2. At first glance, the attacker only gains knowledge of the access pattern of the executing application and this type of attack does not seem severe. However, by running the known algorithm multiple times using different nonces and keys, it is possible to determine the values of the secret keys used in a computation based on the access pattern of the application itself. This makes it possible to extract keys from the executing software. An example of such an attack is presented by Bonneau and Mironov where it was used to retrieve AES keys from OpenSSL implementations [27].

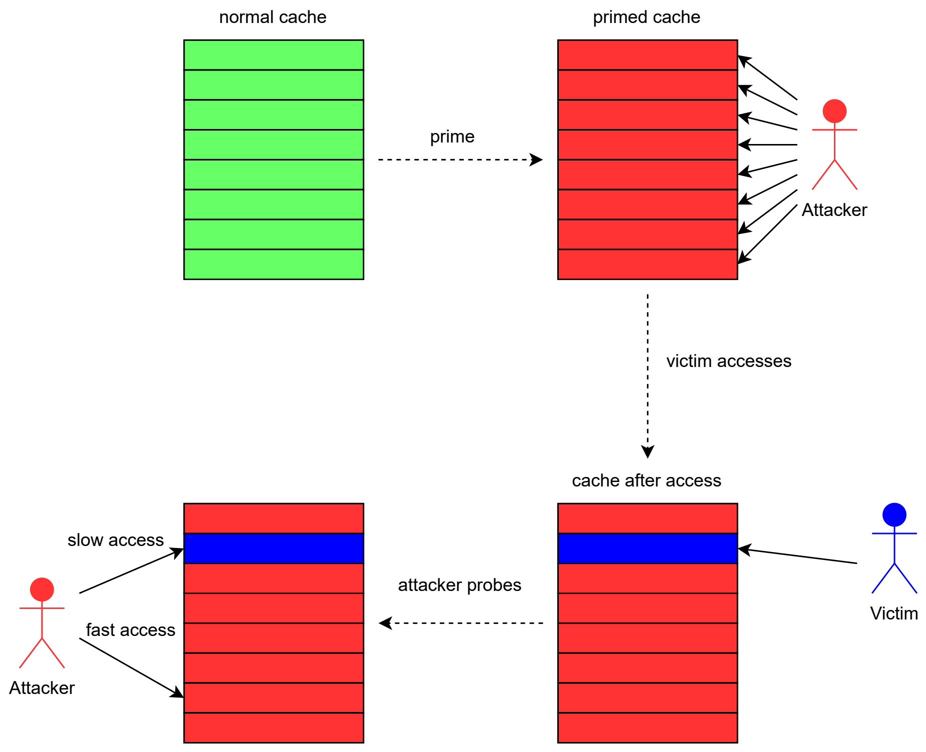

Figure 4.2: prime and probe attack

4.3.6 Side Channel Attacks

Another very prominent attack concerning TEE are the side channel attacks. The attacks under this category use knowledge about the implementation of a system and its side effects. Cache timing attacks discussed in previous sections are one of the most prominent side channel attacks. Power analysis attack is also a type of Side Channel Attack, but at the same time also a type of Physical Attack. Since the attacks of this category already fall into previous categories, they are not handled separately in this thesis.

4.4 Comparison

This section begins by presenting some attacks that were carried out on the respective TEE in the past. The attacks and security measures presented here are taken partly from other papers about this topic [28][22][12]. This section concludes by comparing how the two TEEs performed.

4.4.1 Intel SGX attacks

Intel SGX provides many different security properties, ranging from special processor modes to secure memory areas. Due to many details about the implementation and hardware of Intel SGX being withheld from the general public, this thesis is only going to present some assumptions about potential vulnerabilities and attacks that happened in the past but are now patched by Intel.
Intel SGX, similar to many other TEE, does not emphasize protecting the enclave against physical attacks. There are two reasons for this. First, Intel considers this class of attack to be too expensive to become a real threat. Such attacks require the attacker to have physical access to the device to carry out an attack on it, which is unrealistic for most attackers. Even if they have access to the device, the attacker needs special tools to carry out the attacks, for example, an iCEstick for bus tapping attacks or ion-beam microscopy in the case of chip imaging attacks. This is much more expensive compared to normal software attacks. Second, protecting against these sorts of attacks is costly, as they require expensive design decisions on the architectural level. As a result, there exist several potential physical attacks on Intel SGX. One example is an attack targeting the uncore ring bus in combination with the Generic Debug eXternal Connection (GDXC). The GDXC is a debug port present on Intel architectures, which reports CPU data to an external debugger. This data includes the data transferred in the uncore ring bus. As this information is unencrypted in the SGX architecture an attacker can use the GDXC port to extract the data. Another kind of physical attack that works on Intel SGX are power analysis attacks. As already mentioned, Intel has made no attempt to prevent this type of attack. It should be mentioned that at the time of this writing, no known power analysis attack can be successfully carried out on Intel SGX. However, Intel SGX is not vulnerable to all physical attacks, e.g., bus tapping attacks do not work with SGX. This is because Intel SGX considers the system bus as untrusted, therefore any information transmitted through it is encrypted, in contrast to the uncore bus where the data is unencrypted. Because of this, the data extracted using a bus attack is unreadable for the attacker.
One of the main architectural features of Intel SGX is that it considers the system software as untrusted. This is because system software possesses many vulnerabilities in its large codebase, with most of them unknown to the general public. Intel SGX is very capable of resisting privileged software attacks. The first component preventing malicious system software from gaining access to SGX is the PRM. It is not directly accessible to the system software except for a select few instructions required to maintain it. The second reason is that applications can enter and execute in the enclave mode provided by Intel SGX. The enclave mode is further protected by microcode running inside the processor. This protects the applications from more privileged applications, as they are unable to interfere with it. With all these security measures in place, Intel SGX is capable of protecting against most direct privileged software attacks, but there are still some vulnerabilities in place. One of them is the fact that while the applications execute inside the enclave mode, the management of its memory and resources is done by the untrusted OS. This can lead to denial of service attacks if the OS refuses to allocate any resources to the enclave.

As Intel SGX uses the same address translation process as the OS which was described in Section 2.1.3, it is prone to address translation attacks described in Section 4.3.4. It protects itself from these attacks by storing the virtual address of each page in the EPCM, which prevents the active address translation attacks described earlier, even if they are evicted by the OS which it is allowed to do. Even memory mapping attacks based on TLB are not possible, since every time the processor exits the enclave mode it completely flushes the TLB. Additionally, when storing new entries into the TLB Intel SGX performs security checks to prevent abnormal behaviours. Passive address translation attacks are still possible. The OS is responsible for part of the address translation process used by Intel SGX and manages its memory. It is therefore possible for it to monitor access patterns of the software running inside the enclave.
Software attacks on peripherals aim to use connected devices to gain memory access or alter features to enable more potent attacks. As Intel SGX configures the memory controller to reject any attempts at accessing memory within the PRM, attacks attempting to access memory are prevented. Even the rowhammer attack mentioned in Section 4.3.4 is unable to harm Intel SGX integrity. This is because the memory is protected by cryptographic primitives which are stored in the memory encryption engine and check the integrity of the memory regularly. Therefore, if a bitflip caused by a rowhammer attack were to occur, it would be detected by the integrity check, causing the attack to fail. The debugging features provided by the processor are not a threat either, as these are disabled while the processor is in enclave mode, assuming the debugging feature is not provided by Intel SGX itself.
The last class of attacks described previously is the cache timing attack. Intel does not attempt to prevent these attacks, as they consider them to be highly complex and expensive. Such attacks require advanced hardware which is inaccessible to the general public. Therefore, it is still possible for malicious OS to make cache timing attacks. One such example is presented by Schwarz [29]. In it, he uses the prime and probe approach described in Section 4.3.5 to extract the RSA key of an RSA computation running inside an enclave. It should be mentioned that this attack is executed by a compromised SGX enclave. It has a success rate of 96 percent.

4.4.2 ARM TrustZone attacks

For the attacks on ARM TrustZone, this thesis wants to preface that the success of every attack heavily relies on the actual realization of TrustZone on the device in question. For example, the realization of secure and nonsecure memory can heavily influence the success of a rowhammer attack. As the rowhammer attack can only cause bitflips in adjacent memory, if the secure memory were to be on a different storage device than the nonsecure memory then the rowhammer attack would inevitably be not possible. This is because the attack is not able to cause bitflips on a different memory device. For this reason, this section will mention the actual system the attacks were carried out against, as the same attack may not be possible against a different ARM TrustZone implementation.
Similar to Intel SGX, ARM TrustZone does not include extra safety measures against physical attacks. Consequently, there are many possible physical attacks against ARM TrustZone, including power analysis attacks, chip imaging attacks, and, depending on the realization of ARM TrustZone, port attacks. One example of a successful port attack against TrustZone is carried out by Benhani [30]. The attack uses a general purpose port included on the Cortex-A based Xilinxs Zynq-7010 platform that connects the TrustZone to a custom built device. This attack allows the attacker to change the value of the NS bit, enabling unauthorized access to the secure world. It should be

mentioned that this port attack may not be possible on other ARM TrustZone platforms, as in most cases the debug ports are disabled after the manufacturing process.
Privileged Software attacks are mostly unfeasible against ARM TrustZone. ARM TrustZone's main design concern is preventing this form of attack. The untrusted system software executing at ring 0 is mostly unaware of the existence of the TEE based on TrustZone. Only a few selected controllers are TrustZone aware. They prevent the OS from accessing memory or other devices that are only accessible while in the secure world and are managed by trusted system software. Therefore, the untrusted OS is unable to access or even monitor the execution happening in the TEE.
Address translation attacks are similarly impossible in the TrustZone architecture. Direct memory access attempts are rejected while the processor is executing in the secure world. This makes any attacks based on Direct Memory Access (DMA) unfeasible. Attacks that attempt to rearrange the memory structure in such a way to facilitate unintended behaviour as described in Section 4.3.4 are also not possible, as those types of attacks require the untrusted OS to manage the memory. This is not the case in ARM TrustZone as memory in the secure world is managed by the trusted system software running in it. Similarly, attacks attempting to abuse a stale TLB are out of the question, as the TLB is also managed by the secure OS. In contrast to Intel SGX, passive memory attacks are also impossible, since the untrusted OS is unable to monitor memory access patterns in the secure world.
Rowhammer attacks, a prominent attack on a peripheral, success depends on the actual implementation of ARM TrustZone. As rowhammer attacks aim to cause bit flips in neighboring memory rows, they are only possible if non-secure memory is adjacent to an important memory row that is part of the secure world. If the secure memory is on a different memory storing device, then rowhammer attacks are not possible. If the memory part of the secure world is stored on memory devices shared with the normal world, then the designers of the processors have to add additional integrity checks to protect against rowhammer attacks, as ARM TrustZone itself does not include them. One example of a successful rowhammer attacker is shown by Carru [31]. The attack used a bitflip to circumvent prevention measures and gain access to secure memory. From there, the attacker was able to extract RSA keys, breaking the secure storage provided by ARM TrustZone.
Cache timing attacks are feasible to attack ARM TrustZone. This is because even though the cache lines are partitioned into nonsecure lines and secure lines, the allocation happens at runtime. This enables nonsecure processors to use all cache lines, paving the way for cache timing attacks as it is still possible for them to manipulate the cache lines in a way to gain information about the access pattern of secure applications. Armageddon presents a cache timing attack based on the prime and probe approach mentioned in Section 4.3.5 [32]. The devices attacked in this paper are based on Qualcomm Snapdragon 801, Qualcomm Snapdragon 410 and Samsung Exynos 7 Octa 7420, but it should also be possible on many other ARM TrustZone implementations. This attack enabled malicious applications without any privileges to steal different types of information, ranging from touchscreen inputs to cryptographic primitives stored in Java. Another attack presented by Zhang is also based on prime and probe [33]. It is able to steal the AES key from an AES encryption scheme running in the secure world without using a malicious OS. Based on these successful attacks, it can be concluded that ARM TrustZone, similarly to Intel SGX, is unable to prevent this class of attack.

4.4.3 Conclusion

The presented attacks show that attacks aimed at memory storage are generally not feasible for both Intel SGX and ARM TrustZone. The fact that the memory storage used by the secure world is stored in the secure world and managed by it in ARM TrustZone is sufficient to ensure secure storage, under the assumption that it is properly protected by memory access controllers. Similarly, the PRM and its security mechanism are sufficient to ensure the feature of secure storage for Intel SGX based TEEs, even though the memory is managed by an untrusted OS. The only attack enabled by this management are passive memory attacks. Neither memory mapping attacks nor DMA based attacks can break the security mechanisms in place for both TEE. Attempting to attack both TEEs through their peripherals is also not possible. Integrity checks, or the complete separation of nonsecure and secure memory, prevent rowhammer attacks from having their desired effects. While it may be able to cause bit flips in the secure memory area for both environments, it is unable to capitalize on it as the integrity checks detect it fast enough to prevent unintended behaviour, assuming these integrity checks exist in the ARM TrustZone implementation. Port attacks are also generally not possible. This is because the debug ports that may be abused to mount these attacks are deactivated shortly after the manufacturing process is done. Both TEEs start to falter when facing more sophisticated attacks. This can be seen by the various side channel attacks that are possible to use against Intel SGX and ARM TrustZone. These attacks include power analysis attacks on the physical level or cache timing attacks on the software level. They can compromise the confidentiality of the current execution happening inside the TEE. These attacks are hard to prevent as they exploit the inherent features of computers, e.g., the time difference between cache misses and hits; or the inherent need for power supply for any electrical hardware. Therefore, fixes are hard to come by, as one can not simply equalize the time needed for a cache miss and hit. Finding measures against side channel attacks is a topic of future research to ensure the continued success of TEEs and computers in general.

The overall conclusion is that the security features both TEEs provide are very similar. Both environments are secure from address translation attacks, privileged software attacks, and attacks on their peripherals, with only Intel SGX being vulnerable to passive memory attacks. At the same time, both TEEs are unable to prevent more advanced attacks, namely, power analysis attacks, chip imaging attacks, and cache timing attacks. Another key factor in their performance, especially regarding ARM TrustZone, is their specific implementation. Depending on the design decisions made during the development process, it could lead to vulnerabilities that can be abused by attackers. This can be seen in Xilinx Zynq-7010 vulnerability to port attacks. Table 4.2 shows the performances of both environments regarding certain attacks.

Table 4.2: Performance of Intel SGX and ARM TrustZone in regards to common attacks

Attack Type	**Intel SGX**	**ARM TrustZone**
physical attacks	Secure for port attacks as debug port disabled, but not secure against other physical attacks as no measures have been taken to prevent them	Secure for port attacks as debug port disabled, but not secure against other physical attacks as no measures have been taken to prevent them
privileged software attacks	Secure as system software is untrusted and unable to access enclave memory through special controllers and integrity checks	Secure as untrusted OS is unaware of the secure world and its devices and memories, with TrustZone aware controllers preventing the OS from accessing them
attacks on peripherals	Secure as integrity checks prevent rowhammer attacks from achieving their desired results	Secure as secure memory can be seperated from non-secure memory, preventing bit flips in secure memory regions through rowhammer, alternatively integrity checks can also be included
address translation attacks	Secure because DMA access attempts are rejected in PRM, additionally the TLB is flushed after exiting enclave mode and evicted pages are encrypted, passive memory attacks are still possible as memory is managed by untrusted OS	Secure because secure world memory with all relevant components, for example the TLB, are managed by trusted system software, with untrusted OS being unaware and unable to access them
cache timing attacks	not secure, as prime and probe attacks are still possible	not secure, as prime and probe attacks are still possible

5 Comparison for common use cases

This chapter presents some common use cases of trusted execution environments. It begins by giving an overview of the use cases. It proceeds by going over the various use cases this thesis will discuss, including Digital Rights Management, malware detection and protection, and SMC. Each section will also give an example implementation of the use case. This chapter concludes with the final comparison of both TEEs.

5.1 Overview

In general, a TEE can be used to execute any application. The problem is that it may not be efficient. Setting up an enclave, entering the secure world, and any checks that result from executing in the isolated execution environments add extra overhead that may not be desirable for common applications. Therefore, TEE are mostly used for use cases that require higher security standards that a common execution environment may not be able to provide. These use cases can range from smaller applications that only aim to protect information, to larger ones where confidential code is executed by themselves or enabled to be executed by third parties.
The first use case to be described is Digital Rights Management. It is one of the most common use cases. It deals with the protection and distribution of digital content: e.g. films, songs, or games. The second topic is mobile transactions. A use case that is becoming more and more important in recent years, as digital payment methods like AliPay becoming more commonplace. Another use case that TEE are used for is mobile identity. This use case deals with anything that involves the authentication of the user using a mobile device. Malware detection is a use case that TEE can also be used for. By having the TEE regularly scan the OS for malware or malicious behaviour, it can be used to improve the reliability of normal system software. More advanced use cases include anonymous attestation, an extension of remote attestation where the attesting party remains anonymous. Another advanced use case is secure multiparty execution, where a piece of code is executed using the inputs of many different parties without revealing anything about the inputs. The final subsection presents verifiable cloud computing, an approach to make cloud computing more secure by protecting the information from potentially malicious cloud providers.
Table 5.1 gives an overview of the final results.

Table 5.1: Performance of Intel SGX and ARM TrustZone in regards to common use cases

Use Case	**Intel SGX**	**ARM TrustZone**
DRM	✓	✓
mobile transactions	✗	✓
mobile identity	✗	✓
Malware Detection/Prevention	✓	✓
Anonymous Attestation	✓	✗
SMC	✓	✗
Verifiable cloud computing	✓	✗

5.2 Digital Rights Managment

One major use case for TEE is DRM. A TEE is capable of enforcing DRM on user devices. By using the TEE to restrict unauthorized access to copyrighted content, for example, movies, songs, or applications, companies can distribute these while avoiding legal concerns. One common approach to DRM using TEE is to first distribute encrypted content, for example, an encrypted movie or game for pre-download. The user can download the movie but is initially unable to play it. If he wants to watch it the device must first access a licensing server to get an access key. The licensing server assigns an access key if the user has a valid license or payment. A TEE then uses the provided key to decrypt the movie so that the user can watch it. It should be mentioned that the decrypted version of the movie is stored inside the TEE and the user is unable to extract it to distribute it illegally to other people. This enables different monetization schemes. For example, the user may only borrow the movie for a week. After a week passed the TEE would then block the user's attempt to access the content. Additionally, the TEE can be used to protect against the stealing of copyrighted content, for example, a movie could be recorded using screen recorder software and illegally redistributed. The TEE can prevent this by having privileged access to the output device and preventing other applications from having access to it. One example of a DRM application is PlayReady [34]. It is a DRM solution developed by Microsoft that manages content protection and distribution. It is divided into multiple security levels: SL150, SL2000, and SL3000. SL150 is the lowest security level and is unable to enforce any security guarantees. SL3000 is the highest which uses a TEE implementation to protect important information, such as the decrypted content, the key used for the decryption, and any licensing information.

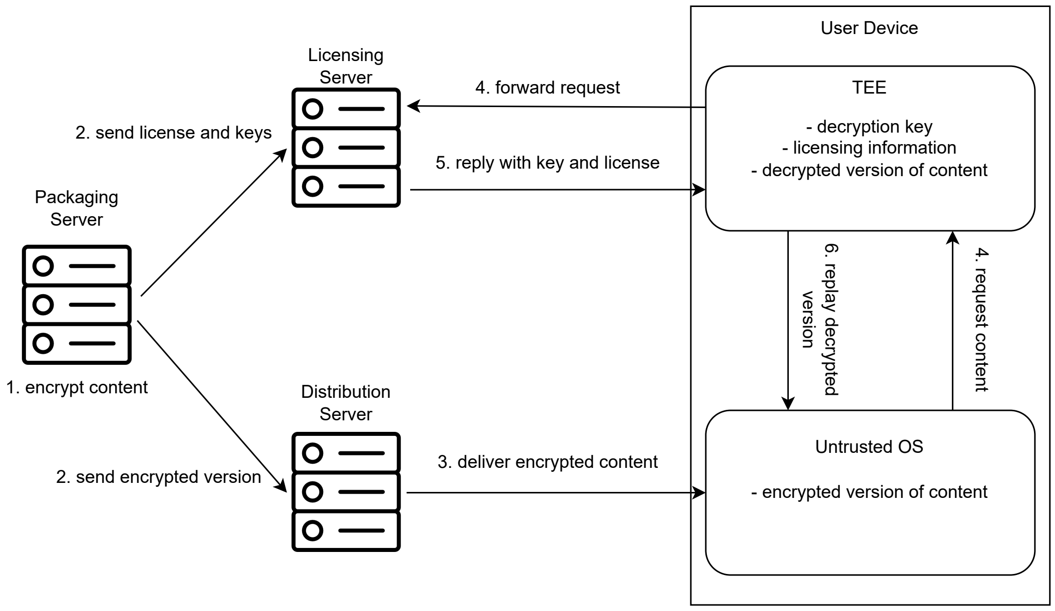

Figure 5.1: PlayReady workflow

The architecture and workflow of PlayReady are depicted in Figure 5.1. It starts by having the content in question get encrypted. This happens at a packaging server that then proceeds to send the encrypted content to the distribution server. The encryption key and the license information are messaged to the license server. When a user wants to use copyrighted content, the TEE on the device sends a request to the license server that contains relevant information about the user and the content the user wants to consume. After the licensing server confirms the request it answers by providing the TEE with the decryption key and license information. The TEE can then use the provided key to decrypt the requested content and let the user consume it as defined by the licensing agreement.
PlayReady can be implemented using both Intel SGX and ARM TrustZone.

5.3 Mobile Transactions

Another prominent use case for TEE are mobile transactions. With the increasing use of mobile payment applications such as GooglePay, PayPal, or WePay modern smartphones have to achieve more stringent security measurements. This is because mobile payment applications use important security relevant credentials and have a wide array of transaction methods, ranging from NFC, QR Codes to in-app purchases. Therefore, these applications have a big attack surface. Since the credentials of the parties involved in the transaction are transmitted across different participants, their protection is extremely important. TEE can be used to secure these credentials in transit and at rest.
One prominent application that uses TEE to enable mobile transactions is Samsung Pay [35]. It can be used for a wide range of use cases. Examples include peer to peer transactions and in-app payments. Samsung Pay uses Samsung KNOX technology, a software TEE that uses ARM TrustZone to create a secure storage and an isolated execution environment. Samsung Pay uses

important credentials, for example, card information, to create a secure token that is stored in the secure storage provided by Samsung KNOX. This token does not contain any information about the card but can be used by retailers for transaction purposes. The workflow of this system can be summarized as follows. When a user wants to initiate a payment the user starts by authenticating himself through passcode or biometrics. After that, the trusted application running inside the secure world will use trusted drivers to pass the secure token to the retailer. The retailer can then use the secure token to finalize the transaction. This approach keeps the credentials of the user opaque for the retailer. Figure 5.2 shows the structure of Samsung Pay on a Samsung Smartphone.
Samsung Pay is implemented using Samsung KNOX, which is implemented on ARM TrustZone.

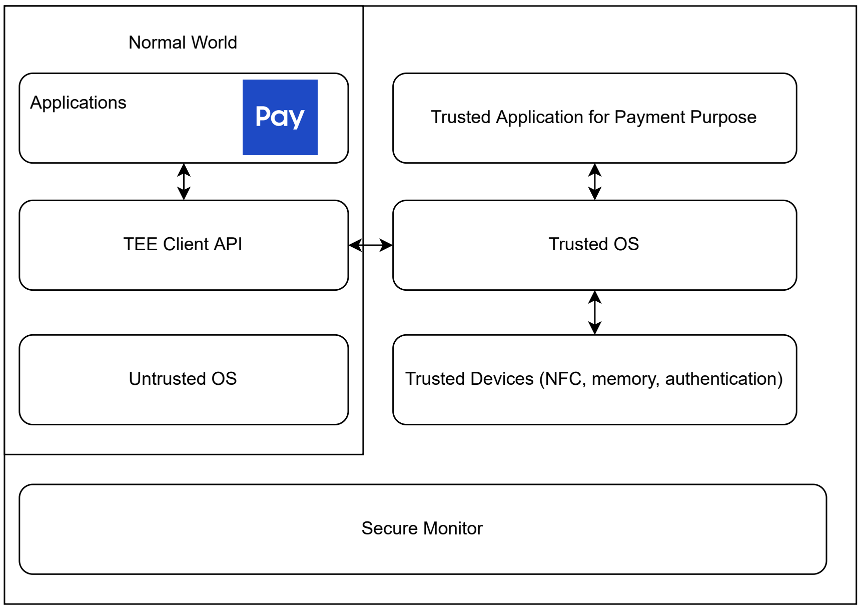
Normal World
Applications
Pay
Trusted Application for Payment Purpose
TEE Client API
Trusted OS
Untrusted OS
Trusted Devices (NFC, memory, authentication)
Secure Monitor

Figure 5.2: Samsung Pay structure

5.4 Mobile Identity

Mobile Identity is another use case for TEE on mobile platforms. Mobile Identity describes any application with some kind of secure data in it. Such application can be used to authenticate the user, for example by using a smartphone application to replace a driver's license or a passport. Another example is that during the coronavirus epidemic the application CovPass is used as a digital vaccination certification. These information contain private interests critical to most users. Their protection on a potentially compromised device is therefore vital. This can be seen in a recently passed directive by the European Union, the Payment Services Directive. The directive requires companies to use strong customer authentication and even recommends using secure execution environments in the regulatory technical standards section of the law [36]. Even though the secure execution environment could be implemented using only software methods, hardware solutions like a TEE based on ARM TrustZone are a more secure option. More specifically an application could use the secure storage provided by a TEE to store important information and communicate it to interested parties using a trusted path to an output device, e.g., through NFC communication or the display of the smartphone.

One example application is HYPR [37]. It is a passwordless multi factor authentication application used by many companies. It allows the pairing of mobile devices with devices such as workstations and authenticating the user without passwords. This is because passwords are considered error prone. Instead, HYPR uses biometrics, PIN or face unlock. These features are provided by most smartphone devices to unlock computers or similar. To achieve this, they run part of the application in the secure world provided by ARM TrustZone, which is responsible for the authentication process and stores security relevant information in the storage of the secure world. Figure 5.3 shows how this application runs on a mobile device using ARM TrustZone.
As shown by HYPR, ARM TrustZone is the preferred TEE when it comes to mobile identity. The main reason is that mobile identity prefers smaller devices that are easier to carry around. Most smaller devices use processors based on the ARM architecture.

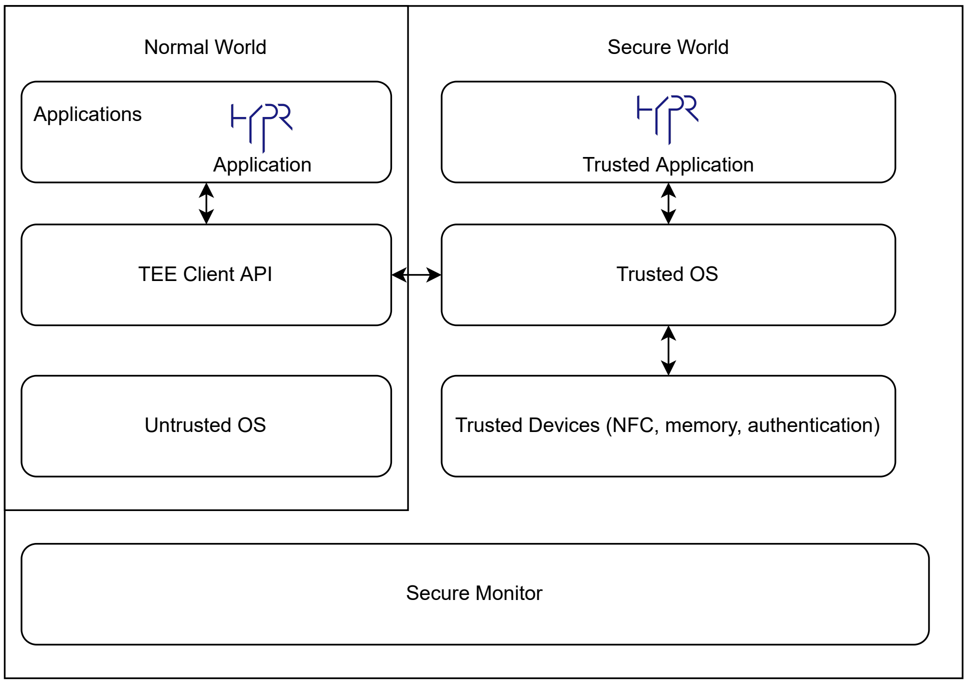

Figure 5.3: HYPR structure

5.5 Malware Detection and Prevention

With the increasing prominence of malware attacks that put user data at risk, common solutions like behaviour based malware detection are quickly becoming insufficient. Especially ransomware attacks that encrypt user data and use a direct monetization model are becoming more rampant and can have a high impact on the victim. One possible solution for this problem is the use of TEE. By storing important user data in the secure storage of a TEE it is possible to prevent malware from altering those. One such application is presented by Zhao where a TEE is used to securely store important user data[38]. Another more prominent implementation of malware detection is done by Samsung Knox, with a feature called Knox Attestation [39]. In KNOX Attestation, the TEE is used to check the health of the device, for example, whether it has been tampered with, rooted, or has malware. This information is then bound to a nonce and signed with an attestation key. The key can then be verified by Samsung's Attestation Server. This enables the TEE to serve as a root of trust for the device, improving its trustability. Figure 5.4 illustrates the workflow of KNOX attestation.

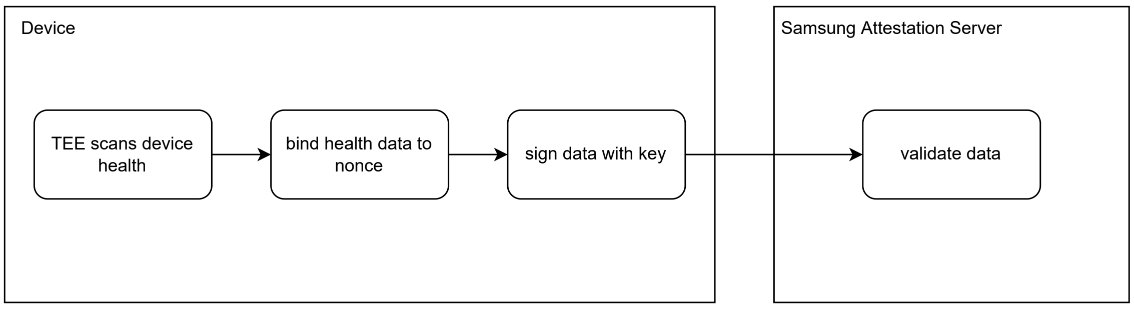

Figure 5.4: KNOX Attestation workflow

In regard to Malware Detection and Prevention, both Intel SGX and ARM TrustZone can be used, as it mainly relies on having a secure area to verify the health of the main system software.

5.6 Anonymous attestation

Anonymous attestation is an extension of remote attestation. Remote attestation confirms to a third party that a TEE is being run on a computer and that its integrity has not been compromised. Anonymous attestation adds on to remote attestation that the user of the TEE remains anonymous. I.e. it only exposes that a TEE is used, but not which one is used. A simple solution to this would be to use an asymmetric encryption scheme, where each TEE has the same secret key. The TEE could then authenticate itself to a third party using the secret key. And because each secret key is the same the verifier can not identify the TEE. However, this approach has major drawbacks. If a single TEE is compromised or the secret key is somehow leaked, the attestation scheme would become unusable. This raises another major challenge for anonymous attestation schemes, it must have the ability to detect revoked clients, without actually knowing who the client is.
The first solution for anonymous attestation was presented in 2004 by Brickell, Camenish, and Chen [40]. It was initially developed for the use of TPM. It achieved this by using a group signature with the verifier being unable to identify the specific signer of a signature. The problem with this scheme is that it is very hard to revoke a compromised client. This solution was therefore improved later on in 2007, by Brickell and Li, to ease the revocation of certain secret keys, called Enhanced Privacy ID (EPID) [41]. While this was possible in the old scheme, it required the secret key of the compromised TPM to be published. This is usually not the case since the attacker would most likely not publish the key but use the compromised TPM to perform further attacks. EPID works around this by being able to revoke a system with only an example signature of a compromised system, without the need to know the underlying secret key. EPID is nowadays used by Intel in its remote attestation scheme described in Section 3.2.3. It should be indicated that Swami argues in his paper that the EPID scheme used by Intel SGX does not provide anonymity [42]. This is because Intel SGXs remote attestation scheme uses a root provisioning key to authenticate a TEE using the Intel Attestation Service. This requires Intel to store the root provisioning key of each manufactured TEE. This effectively destroys the anonymity of the user since Intel is able to identify the TEE currently

running the remote attestation scheme.
ARM TrustZone itself does not inherently support remote attestation, even less anonymous attestation. This does not mean that ARM TrustZone is incapable of including these features. This can be seen in Section 3.1.3, where a remote attestation scheme based on TrustZone is presented. For this reason, ARM TrustZone could support anonymous attestation, but it would require the manufacturer of the processor to add additional components to store important information such as the root provisioning key or an equivalent of it. This potentially makes a solution based on ARM TrustZone more expensive than an Intel SGX solution.

5.7 Secure Multiparty Computing

Another interesting use case is SMC, which enables two or more computers to perform a computation together. Initially developed by Yao in 1982 [43], it relied on a cryptographic protocol, called a garbled circuit, to enable two party computations. Garbled circuits can be understood as an encrypted function. This encrypted function takes encrypted inputs, and after executing returns the unencrypted output. The execution is referred to as decrypting. This allows all members of the computation to know the output, with the provided encrypted inputs providing less information about their unencrypted state than the output itself. This was improved by Goldreich to enable multiparty computation using garbled circuits [44]. While this allowed for SMC, it cannot be considered practical as encrypting and decrypting the function is very resource intensive, with real-time performance being unfeasible [45].
A solution to this problem are TEE. By having multiple parties provide their input to multiple TEE and comparing their outputs to detect compromised members it is possible to perform SMC while offloading the cryptography primitives to the security mechanisms of the TEEs. This is possible as TEE guarantee isolated execution, meaning that the user of the computer in question has no access to the inputs of the other parties, assuming these were provided over a secure channel. The paper by Bahmani presents a way in which SMC can be implemented using TEE [46]. This specific example uses Intel SGX but it also mentions that it can be done using other TEEs such as ARM TrustZone. It relies on two of the TEEs core feature, namely isolated execution and remote attestation. It uses them to guarantee that the execution of the code has not been tampered with or leaked to malicious parties. To achieve this the protocol described in the paper starts with a key exchange to establish a secure channel between the participants. After the input has been shared over the secure channel the TEE can execute it and return the output over the same channel. This reduces the complexity of the operation since the client only has to establish the secure channel and send the input. The function is executed inside the TEE. This removes the need for creating garbled circuits and encrypting the inputs and makes the process much more efficient. As Intel SGX already supports all the required features: isolated execution and remote attestation. This protocol is easier to implement using it. ARM TrustZone supports isolated execution but does not inherently provide remote attestation. Therefore, it is harder to implement but not impossible as it is possible for TrustZone to enable remote attestation.

5.8 Verifiable cloud computing

Cloud computing describes the outsourcing of several resources, for example, storage or computing power and providing them on demand. It is a rising trend as the IoT is becoming more and more prevalent in modern society. The IoT requires a lot of sensors and other small devices to monitor the human world to interconnect the life of its inhabitants a shared resource pool of resources is becoming more and more important. This is because most of the small devices do not have the computation power to process the information in an effective fashion. Cloud computing offers a solution by providing a large amount of computational power on demand, with important features such as scalability, fault tolerance, and pay-per-use. However, this new technology presents new security risks, as outsourcing computations may leak information to the computer executing it. This puts the integrity and confidentiality requirements of users at a potential risk. TEE may present a solution to this problem by providing an isolated execution environment in which the integrity and confidentiality can be verified by the consumer. It is the so called verifiable cloud computing.

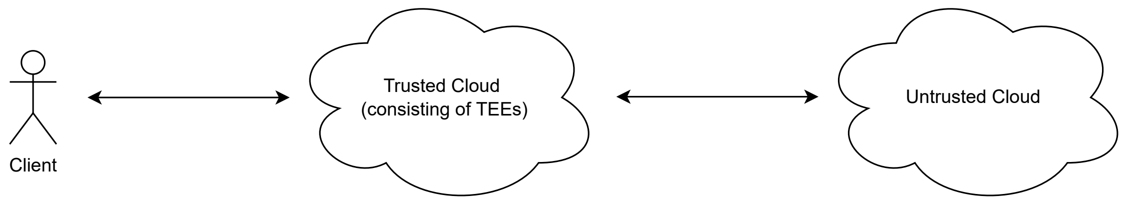

Figure 5.5: Twin Clouds model

One approach for verifiable cloud computing is called twin clouds, outlined by Bugiel, Nürnberger, Sadeghi, and Schneider in their paper [47]. It splits the cloud into two parts. One is the commodity cloud where the majority of the performance intensive execution is made; the second one is called the trusted cloud, where the security critical executions are carried out. This is depicted in Figure 5.5. The trusted cloud can consist of multiple secure hardware, for example, TEEs. They are used to encrypt the functionalities and inputs provided by the client over a secure channel using a version of the garbled circuits described earlier. These encrypted circuits and inputs are then provided to the commodity cloud to execute. This outsources the resource intensive part of the process, making this approach much more scalable. The output calculated by the untrusted commodity cloud is then verified by the trusted cloud and returned to the client. This solution requires the trusted cloud and its parts to be able to verify their integrity. Therefore, remote attestation and isolated execution have to be provided. As Intel SGX already provides these, it is more efficient to use for this approach than ARM TrustZone, because it does not provide remote attestation.

5.9 Comparison

As discussed previously, both TEEs can be used for common use cases. There are many reasons for this. The first is that both Intel SGX and ARM TrustZone were built to realize the same thing, a TEE that provides a secure execution environment on the hardware level with higher security guarantees than the normal OS can provide. Secondly, both TEE can be used for most use cases is that they are often used as hardware TEE enablers, and not as a TEE itself. This applies especially to ARM TrustZone. Many smartphone applications use a software TEE built on top of the hardware TEE features provided by ARM TrustZone as the actual environment. Prominent software TEE build on TrustZone are Google Trusty [48] and Samsung KNOX [49]. KNOX is approved by the US government and is used by its employees. Therefore, the fact that some key features are missing from Intel SGX or ARM TrustZone is not important, as they can be supplemented when developing a software TEE based on them.
Because of the aforementioned reasons the choice of the TEE for a use case is only partially dependent on its features. Instead, the main reason why Intel SGX or ARM TrustZone may be preferred is the device or more specifically the processor of the device. Intel processors are more commonplace in larger devices, such as computers, laptops, or servers. Therefore, use cases that are executed on these platforms like verified cloud computing may prefer to use Intel SGX. If the use case instead requires smaller devices that are easier to carry around like smartphones or IoT devices, then they may prefer ARM TrustZone, as it is more commonplace than Intel SGX on those platforms. Examples of such use cases are mobile transactions and mobile identity. This does not mean that the other TEE is never used by the devices in question, there exist servers and laptops with processors based on the ARM architecture, there are also smartphones that use Intel chips. Of course, they are much rarer.
Therefore, this thesis concludes that both Intel SGX and ARM TrustZone can be used for all use cases discussed in this thesis, and the reason one may be preferred over the other is not necessarily because of the TEE itself but instead the processor they use.

6 Conclusion

This thesis started by introducing core foundations that are required to understand trusted execution environments. It proceeded to discuss their important features: secure storage, isolated execution, remote attestation, secure boot, and trusted path. The thesis then goes over two environments: Intel SGX and ARM TrustZone. The end goal of both environments was the same, enabling an execution environment isolated from the untrusted system software. The approaches Intel and ARM took to achieve that goal were vastly different. Intel SGX executes at ring 3, meaning that most of the resource management is done by the OS. As the OS is considered untrusted, most of the security guarantees in the SGX architecture are based on checks and restrictions imposed on the OS. These are microcode for integrity checks and a careful selection of instructions that can be used by the untrusted OS to interact with security-sensitive resources. ARM TrustZone in contrast executes at ring -2. Therefore, the secure world executes at a higher privilege level than the untrusted OS. This has the advantage that it is not that important to control the behaviour of the system software as it is unable to interact with the secure world. Additionally, the concrete design of the secure world can be left to the manufacturers of the processor. They can freely include extra memory, I/O devices, or other features to improve the functionality of the secure world.
Despite these architectural differences, the security guarantees both TEE offer are largely the same, with both environments being able to resist the same common security threats. Both TEEs consider privileged software attacks as the main attack type, against which both environments perform well. The main vulnerabilities both TEE share mostly stem from the side effects that all computers have. A typical example is the different cache response time depending on whether a piece of memory is allocated or not. Another problem is that preventing sophisticated attacks on the physical level is too expensive, even impossible with the current technology. Therefore, Intel and ARM did not consider these attack vectors when building TrustZone and SGX respectively and left them partially vulnerable to this class of attack.
Regarding common use cases, both TEE aim to provide a secure execution environment for high security applications. Therefore, both environments can be used for all use cases. The reasons why someone may prefer one over the other depends on two aspects. The first is whether the TEE already supports required features. One example of this is anonymous attestation to a third party. Because Intel SGX already uses an anonymous attestation scheme in EPID, it is preferable compared to ARM TrustZone. The second is the device that the use case is run on. The devices ARM TrustZone and Intel SGX are used on have a small overlap. ARM TrustZone is mostly seen on smartphones and IoT devices while Intel SGX is more common on larger laptops or servers. This influences the decision for some use cases, for example when deciding which TEE to use for mobile identity, ARM TrustZone has a large advantage as most devices used for mobile identity use TrustZone.
Overall this thesis concludes that the performance of both Intel SGX and ARM TrustZone is similar. The differences between them on a high level are minor. Their security guarantees are also very similar, both sharing vulnerabilities against side channel attacks. The most important difference regarding security is that Intel SGX is vulnerable to passive attacks as its resources are managed

by the untrusted OS. In terms of use cases, the decision depends on the device and the features required for the use case. In the end, both TEE can be used for all use cases as extending their features through software TEE is common practice in the market.

Bibliography

[1] *Mobile Payment Statistics*. https://www.businessofapps.com/data/mobile-payments-app-market/. Accessed: 2022-03-07 (cit. on p. 13).

[2] I. Arikpo, F. Ogban, I. Eteng. “Von neumann architecture and modern computers”. In: *Global Journal of Mathematical Sciences* 6.2 (2007), pp. 97–103 (cit. on p. 15).

[3] M. Portnoy. *Virtualization essentials*. Vol. 19. John Wiley & Sons, 2012 (cit. on p. 16).

[4] *Wikipedia Protection Ring*. https://en.wikipedia.org/wiki/Protection_ring. Accessed: 2022-03-07 (cit. on p. 17).

[5] Bruce L Jacob and Trevor N Mudge. *Virtual memory: Issues of implementation*. June 1998. URL: http://hdl.handle.net/1903/7466 (cit. on p. 18).

[6] M. Sabt, M. Achemlal, A. Bouabdallah. “Trusted Execution Environment: What It is, and What It is Not”. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. 2015, pp. 57–64. DOI: 10.1109/Trustcom.2015.357 (cit. on p. 20).

[7] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, J. M. McCune. “Trustworthy execution on mobile devices: What security properties can my mobile platform give me?” In: *International conference on trust and trustworthy computing*. Springer. 2012, pp. 159–178 (cit. on p. 20).

[8] Z. Ning, F. Zhang, W. Shi, W. Shi. “Position Paper: Challenges Towards Securing Hardware-Assisted Execution Environments”. In: *Proceedings of the Hardware and Architectural Support for Security and Privacy*. HASP ’17. Toronto, ON, Canada: Association for Computing Machinery, 2017. ISBN: 9781450352666. DOI: 10.1145/3092627.3092633. URL: https://doi.org/10.1145/3092627.3092633 (cit. on p. 21).

[9] D. Champagne, R. Lee. “Scalable architectural support for trusted software”. In: Jan. 2010, pp. 1–12. DOI: 10.1109/HPCA.2010.5416657 (cit. on pp. 21, 24).

[10] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, S. Devadas. “AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing”. In: *Proceedings of the 17th Annual International Conference on Supercomputing*. ICS ’03. San Francisco, CA, USA: Association for Computing Machinery, 2003, pp. 160–171. ISBN: 1581137338. DOI: 10.1145/782814.782838. URL: https://doi.org/10.1145/782814.782838 (cit. on pp. 21, 24).

[11] V. Costan, I. Lebedev, S. Devadas. “Sanctum: Minimal hardware extensions for strong software isolation”. In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, pp. 857–874 (cit. on p. 24).

[12] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer’s Manual,* 2021. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html (cit. on pp. 24, 32, 47).

[13] *Information technology — Trusted platform module library*. Norm. 2015 (cit. on p. 24).

[14] X. Ruan. *Platform Embedded Security Technology Revealed*. Springer Nature, 2014 (cit. on p. 25).

[15] *Intel popularity*. https://www.cpubenchmark.net/market_share.html. Accessed: 2022-02-20 (cit. on p. 25).

[16] *ARM processor popularity*. https://www.statista.com/statistics/1132112/arm-market-share-targets/. Accessed: 2022-02-20 (cit. on p. 25).

[17] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, S. Martin. "Trustzone explained: Architectural features and use cases". In: *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE. 2016, pp. 445–451 (cit. on p. 27).

[18] S. Pinto, N. Santos. "Demystifying arm trustzone: A comprehensive survey". In: *ACM Computing Surveys (CSUR)* 51.6 (2019), pp. 1–36 (cit. on p. 27).

[19] Silicon Labs. *Which ARM Cortex Core Is Right for Your Application: A, R or M?* 2020. URL: https://www.silabs.com/documents/public/white-papers/Which-ARM-Cortex-Core-Is-Right-for-Your-Application.pdf (cit. on pp. 27, 28).

[20] W. Ziwang, Y. Zhuang, Z. Yan. "TZ-MRAS: A Remote Attestation Scheme for the Mobile Terminal Based on ARM TrustZone". In: *Security and Communication Networks* 2020 (Sept. 2020), pp. 1–16. DOI: 10.1155/2020/1756130 (cit. on p. 31).

[21] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon, et al. "{fTPM}: A {Software-Only} Implementation of a {TPM} Chip". In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, pp. 841–856 (cit. on p. 31).

[22] V. Costan, S. Devadas. "Intel sgx explained." In: *IACR Cryptol. ePrint Arch.* 2016.86 (2016), pp. 1–118 (cit. on pp. 32, 43, 47).

[23] P. Gallagher, A. Director. "Secure hash standard (shs)". In: *FIPS PUB* 180 (1995), p. 183 (cit. on p. 37).

[24] H. Krawczyk. "SIGMA: The 'SIGn-and-MAc'approach to authenticated Diffie-Hellman and its use in the IKE protocols". In: *Annual International Cryptology Conference*. Springer. 2003, pp. 400–425 (cit. on p. 37).

[25] R. Wojtczuk, J. Rutkowska. "Attacking SMM memory via Intel CPU cache poisoning". In: *Invisible Things Lab* (2009), pp. 16–18 (cit. on p. 43).

[26] O. Mutlu, J. S. Kim. "RowHammer: A Retrospective". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.8 (2020), pp. 1555–1571. DOI: 10.1109/TCAD.2019.2915318 (cit. on p. 44).

[27] J. Bonneau, I. Mironov. "Cache-Collision Timing Attacks Against AES". In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Ed. by L. Goubin, M. Matsui. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 201–215. ISBN: 978-3-540-46561-4 (cit. on p. 45).

[28] M. A. Mukhtar, M. K. Bhatti, 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)*. 2019, pp. 299–304. DOI: 10.1109/C-CODE.2019.8680982 (cit. on p. 47).

[29] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, S. Mangard. “Malware guard extension: Using SGX to conceal cache attacks”. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2017, pp. 3–24 (cit. on p. 48).

[30] E. M. Benhani, C. Marchand, A. Aubert, L. Bossuet. “On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC”. In: *2017 30th IEEE International System-on-Chip Conference (SOCC)* (2017), pp. 108–113 (cit. on p. 48).

[31] *Carru P., Attack trustzone with rowhammer*. https://manualzz.com/doc/36320156/attack-trustzone-with-rowhamme. Accessed: 2022-03-03 (cit. on p. 49).

[32] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, S. Mangard. “ARMageddon: Cache Attacks on Mobile Devices”. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 549–564. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp (cit. on p. 49).

[33] N. Zhang, K. Sun, D. Shands, W. Lou, Y. T. Hou. “TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices”. In: *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 980 (cit. on p. 49).

[34] *PlayReady*. https://docs.microsoft.com/en-us/windows/uwp/audio-video-camera/hardware-drm. Accessed: 2022-02-21 (cit. on p. 54).

[35] *Samsung Pay*. https://pay.samsung.com/developers. Accessed: 2022-02-21 (cit. on p. 55).

[36] *PSD2*. https://ec.europa.eu/commission/presscorner/detail/en/MEMO_17_4961. Accessed: 2022-02-21 (cit. on p. 56).

[37] *HYPR*. https://www.hypr.com/. Accessed: 2022-02-20 (cit. on p. 57).

[38] L. Zhao, M. Mannan. “TEE-aided write protection against privileged data tampering”. In: *arXiv preprint arXiv:1905.10723* (2019) (cit. on p. 57).

[39] *KNOX Attestation*. https://docs.samsungknox.com/dev/knox-attestation/index.htm. Accessed: 2022-03-07 (cit. on p. 57).

[40] E. Brickell, J. Camenisch, L. Chen. “Direct anonymous attestation”. In: *Proceedings of the 11th ACM conference on Computer and communications security*. 2004, pp. 132–145 (cit. on p. 58).

[41] E. Brickell, J. Li. “Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities”. In: *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. 2007, pp. 21–30 (cit. on p. 58).

[42] Y. Swami. “Intel SGX Remote Attestation is not sufficient”. In: *Proc. Black Hat USA*. 2017 (cit. on p. 58).

[43] A. C. Yao. “Protocols for secure computations”. In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, pp. 160–164 (cit. on p. 59).

[44] O. Goldreich. “Secure multi-party computation”. In: *Manuscript. Preliminary version* 78 (1998) (cit. on p. 59).

[45] J. I. Choi, K. R. Butler. “Secure multiparty computation and trusted hardware: Examining adoption challenges and opportunities”. In: *Security and Communication Networks* 2019 (2019) (cit. on p. 59).

[46] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, B. Warinschi. “Secure multiparty computation from SGX”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, pp. 477–497 (cit. on p. 59).

[47] S. Bugiel, S. Nurnberger, A. Sadeghi, T. Schneider. “Twin clouds: An architecture for secure cloud computing”. In: *Workshop on cryptography and security in clouds (WCSC 2011)*. Vol. 1217889. 2011 (cit. on p. 60).

[48] *Google Trusty TEE*. `https://source.android.com/security/trusty`. Accessed: 2022-03-07 (cit. on p. 61).

[49] *Samsung KNOX*. `https://www.samsungknox.com/en`. Accessed: 2022-03-07 (cit. on p. 61).

All links were last followed on March 15, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature