TODO: Replace with approval doc:

https://www.pdx.edu/gradschool/thesis-and-dissertation-information

An Overview of Trusted Execution Environments with a Deep Dive into RISC-V Physical Memory Protection and Hopefully a Better Title

by

Stephano Cetola

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
John Acken, Chair
Roy Kravitz
Tom Schubert

Portland State University
2021

Preface

This thesis tells you all you need to know about...

Acknowledgments

I would like to thank...

Contents

List of Tables

List of Figures

List of Figures

Acronyms

AEX Asynchronous Enclave Exit. 4

AMBA Advanced Microcontroller Bus Architecture. 5

AXI Advanced eXtensible Interface. 5

DMA Direct Memory Access. 3

EPC Enclave Page Cache. 3, 4

MSR model-specific register. 2

OP-TEE Open-source Portable TEE. 5

PRM Processor Reserved Memory. 2, 3

RoT Root of Trust. 2

SGX Software Guard Extensions. 2, 4, 5

SMM System Management Mode. 3

SoC System on Chip. 5, 7

TCB Trusted Compute Base. 2

TF-A Trusted Firmware-A. 5, 6

TPM Trusted Platform Module. 2

TXT Trusted Execution Technology. 2

Chapter 1

Introduction

1.1 Problem Statement

Historically, computer architecture security relied on processor modes or privilege modes where code was allowed to execute. In these modes, separation of privileges is achieved and often referred to as "rings" with "ring 0" being the most privileged (machine mode, kernel code) and ring 3 being the least privileged (user mode, application code). Device drivers run in the rings between these two modes with virtualization being the most common use case[1]. As applications became more complex, specifically with the advent of large-scale virtualization and the internet, this simple security model broke down as executed code could no longer be trusted, nor its origin verified. The problem of "secure remote computation" arises where the data owner must trust not only the software provider, but also the remote computer and infrastructure on which that software is executed. Homomorphic encryption solves this problem to some extent, however the performance overhead of this transaction limits its application [1].

In an attempt to address these issues, microprocessor designers have implemented versions of a "Trusted Execution Environment", first defined by the Open Mobile Terminal Platform and ratified in 2009 [2]. The OMTP standard was transferred to the Wholesale Applications Community (WAC) in 2010 and in July 2012 WAC itself was closed, with the OMTP standards being transferred to The

[1]The virtual machine's kernel, for example, runs in ring 1, one level up from machine mode.

GSM Association (originally Groupe Spécial Mobile)[3]. In this paper we will discuss the two most prevalent implementations of this standard for the x86-64 and AArch64 architectures, as well as a completely open source hardware and software implementation of a TEE for the RISC-V architecture.

1.2 The Intel SGX Solution

Intel Software Guard Extensions (SGX) is built on designs of software attestation already proven in technologies like the Trusted Platform Module (TPM) and Intel Trusted Execution Technology (TXT). In SGX, these concepts of software attestation are used to create containerized sections of memory on the remote computer called "secure enclaves" where data and code can be loaded or executed securely. These enclaves are verified by both a cryptographic attestation key of the container's contents as well as a hardware Root of Trust (RoT) manufacturer's key. Unlike the TPM and TXT technologies, SGX securely operates only on a small amount of data and code called the Trusted Compute Base (TCB), leaving the majority of memory outside this TCB.

1.3 Initial SGX Enclave Setup

Configuration settings for SGX exists as part of the platform firmware, and most firmware vendors provide simple tools for enabling SGX. If SGX is enabled, the firmware is responsible for setting aside a memory region called the Processor Reserved Memory (PRM), and most firmware tools allow specifying the size of the space allocated. The firmware allocates the PRM by setting a pair of model-specific registers (MSRs), collectively known as the PRMRR. The CPU will then protect the PRM from all non-enclave memory accesses including kernel, hypervisor and

System Management Mode (SMM) accesses, as well as Direct Memory Access (DMA) from peripherals [4].

This section of specially allocated memory is used to store the Enclave Page Cache (EPC), which are the 4kb pages holding both the enclave data and code. The exact layout of the PRM and EPC are model-specific, and depend on firmware settings. While untrusted system software both assigns these EPCs to an enclave and loads them with data, it is the CPU which keeps track of all the EPCs ensuring that they only belong to one enclave. Once the system software loads data into the enclave it asks the CPU to mark that enclave as initialized, after which no other data may be loaded into the enclave as this setup process is disabled for that enclave. After initialization, this enclave is measured by a cryptographic hash to ensure that any operations performed on the enclave are done so in a secure environment.

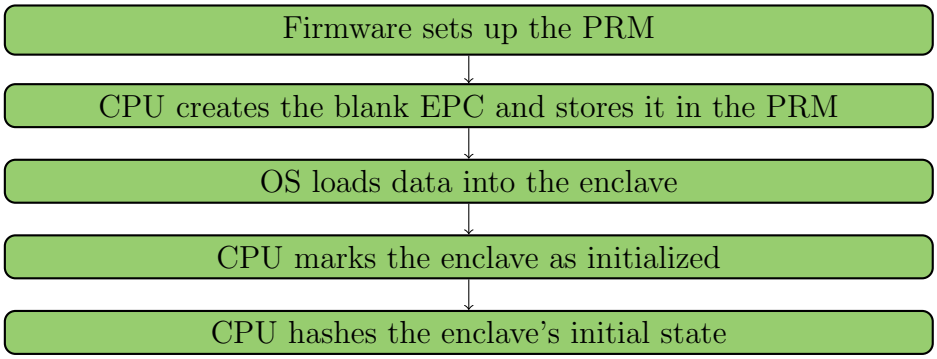

Figure 1.1: **Workflow for setting up an enclave.**

1.4 Executing SGX Enclave Code

Execution flow can only move into an enclave via a special CPU instruction, much like switching from user mode to kernel mode. The actual execution happens in user mode and takes advantage of address translation from the Operating System or hypervisor. The CPU executing the enclave code performs an Asynchronous Enclave Exit (AEX) whenever execution moves outside the enclave such as servicing an interrupt or during a page fault. The CPU state is saved inside the enclave before exiting ensuring that the CPU can security restore the state of execution. There are special machine mode CPU instructions that are used both in allocating EPC pages to the enclave as well as evicting those pages into untrusted DRAM. This facilitates code outside the enclave to operate on code within the enclave. SGX uses cryptographic protections to assure the confidentiality, integrity and freshness of the evicted EPC pages while they are stored in untrusted memory [4]. In this way, Intel SGX is able to allow a specific amount of code and data to remain protected while still allowing access to that data by code outside the trust boundary.

1.5 The Arm TrustZone Solution

When evaluating how Arm's TrustZone works, we must remember several important distinctions. Firstly, the Arm specifications include several different architectures with several different states. Each Arm architecture and state combination may operate slightly differently in regard to how TrustZone is implemented. This paper will only consider the ARMv8-A architecture running in the AArch64 state. Secondly, hardware manufacturers may choose to implement security in

many ways, and with much more flexibility than in Intel platforms. For simplicity's sake, this paper will only cover standard Arm solutions for TrustZone implementation, provided by Arm's Trusted Firmware-A (TF-A) and Open-source Portable TEE (OP-TEE).

Arm System on Chip (SoC) processors create a more absolute separation between the worlds of "secure" and "normal or insecure" operation than Intel SGX. This is accomplished using three principal technologies on the bus, the SoC core, and the debug infrastructure. Firstly, the bus interface, called the Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI), partitions all of the SoC's hardware and software resources by taking advantage of a set of bits. Hardware logic present in this "TrustZone-enabled AMBA3 AXI" bus fabric ensures that no "Secure World" resources can be accessed by "Normal World" components. These bits include AWPROT for write transactions and ARPROT for read transactions (low is Secure and high is Non-secure). Secondly, SoCs using cores like the ARMv8-A include implemented extensions which enable a single physical processor core to safely and efficiently execute code from both the Normal World and the Secure World in a time-sliced fashion. Lastly, the security-aware debug infrastructure controls debug access to the Secure World [5]. These three technologies provide a framework or scaffolding on which to build a platform capable of secure computation.

1.6 Arm Trusted Firmware

Since 2013, Arm has provided Trusted Firmware-A (TF-A) as an open source reference implementation of the firmware required to develop Secure World software for

A-Class devices (including ARMv8-A). The TF-A provides may features including secure device initialization, modular boot flow, trusted boot, and the secure monitor that allows switching between the Normal World and the Secure World. It should be noted that all of this code and documentation is freely available at `https://www.trustedfirmware.org/`. The Trusted Firmware Project is a 'not for profit' open source project hosted by Linaro Limited ("Linaro").

Arm Trusted Firmware uses the scaffolding provided by the A-Class devices to implement the key aspects of TrustZone, namely Trusted Boot and the Secure Monitor. In oder to understand these systems we must first understand the privilege levels available to an Arm A-Class platform.

Privilege Level	**Description**	**Implementation**
EL-0	Application Privilege Level	Supported by CPU architecture
EL-1	Kernel Privilege Level	Supported by CPU architecture
EL-2	Virtualization Privilege Level (Optional)	Supported by CPU architecture
EL-3	Secure Privilege Level	Supported by CPU architecture or a dedicated embedded security processor

Table 1.1: Arm Privilege Level Mapping

Unlike Intel platforms which refer to their privilege levels as rings, Arm uses "Exception Levels" EL0 through EL3 [6].

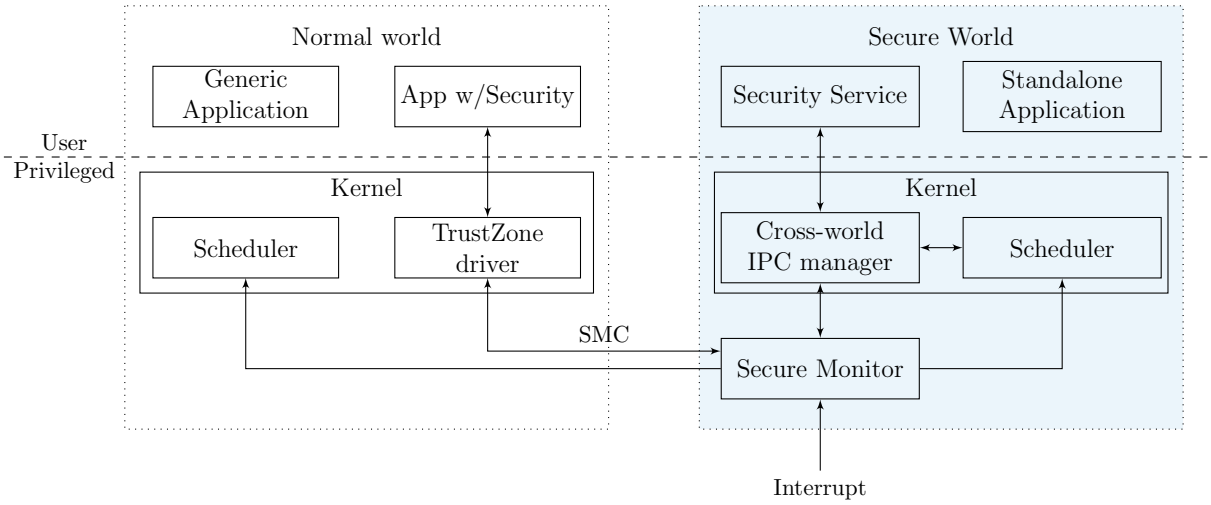

Figure 1.2: **Secure World implementation using ARM TrustZone.** The SoC boots into the Secure World and a monitor is registered which acts as the interface between the Secure and Normal Worlds.

References

[1] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[2] Open Mobile Terminal Platform Group, "Omtp hardware requirements and defragmentation, trusted environment omtp tr1," *Open Mobile Terminal Platform*, 2009.

[3] S. Perez. (2012) Wac whacked: Telecom-backed alliance merges into gsma, assets acquired by api management service apigee. [Online]. Available: https://techcrunch.com/2012/07/17/wac-whacked-telecom-backed-alliance-merges-into-gsma-assets-acquired-by-api-management-service-apigee/

[4] V. Costan and S. Devadas, "Intel SGX Explained," Cryptology ePrint Archive, Report 2016/086, Jan. 2016, http://eprint.iacr.org/2016/086.

[5] Arm Limited, "Arm security technology building a secure system using trustzone technology," 2009. [Online]. Available: https://static.docs.arm.com/genc009492/c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[6] Arm Limited, "Fundamentals of ARMv8-A," 2017. [Online]. Available: https://developer.arm.com/documentation/den0024/a/fundamentals-of-armv8