Secure Processors Part I: 
Background, Taxonomy for 
Secure Enclaves and Intel 
SGX Architecture

Victor Costan, Ilia Lebedev and Srinivas Devadas
victor@costan.us, ilebedev@mit.edu and devadas@mit.edu
Computer Science and Artificial Intelligence Laboratory 
Massachusetts Institute of Technology
Editorial Scope

Topics

Foundations and Trends® in Electronic Design Automation publishes survey and tutorial articles in the following topics:

- System level design
- Behavioral synthesis
- Logic design
- Verification
- Test
- Physical design
- Circuit level design
- Reconfigurable systems
- Analog design
- Embedded software and parallel programming
- Multicore, GPU, FPGA, and heterogeneous systems
- Distributed, networked embedded systems
- Real-time and cyberphysical systems

Information for Librarians

Foundations and Trends® in Electronic Design Automation, 2017, Volume 11, 4 issues. ISSN paper version 1551-3939. ISSN online version 1551-3947. Also available as a combined paper and online subscription.
Secure Processors Part I:
Background, Taxonomy for Secure Enclaves and
Intel SGX Architecture

Victor Costan, Ilia Lebedev and Srinivas Devadas
victor@costan.us, ilebedev@mit.edu and devadas@mit.edu

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
# Contents

1 Introduction 2

1.1 Secure Remote Computation 3
1.2 SGX Lightning Tour 7
1.3 Outline 9

2 A Primer on Computer System Architecture 10

2.1 Overview 11
2.2 Computational Model 13
2.3 Software Privilege Levels 18
2.4 Address Spaces 19
2.5 Address Translation 22
2.6 Execution Contexts 29
2.7 Segment Registers 31
2.8 Privilege Level Switching 34
2.9 An Overview of a Modern Computer System 38
2.10 Out-of-Order and Speculative Execution 44
2.11 Memory Cache Subsystem 49
2.12 Interrupts 62
2.13 Platform Initialization (Booting) 64
2.14 CPU Microcode 69

Full text available at: http://dx.doi.org/10.1561/1000000051
3 A Primer on Security for Trusted Processors 79
3.1 Cryptographic Primitives 80
3.2 Cryptographic Constructs 94
3.3 Software Attestation Overview 101
3.4 Physical Attacks 106
3.5 Privileged Software Attacks 111
3.6 Software Attacks on Peripherals 112
3.7 Address Translation Attacks 117
3.8 Cache Timing Attacks 122

4 A Survey of Secure Processors 128
4.1 The IBM 4765 Secure Coprocessor 128
4.2 ARM TrustZone 132
4.3 The XOM Architecture 135
4.4 The Trusted Platform Module (TPM) 136
4.5 Intel’s Trusted Execution Technology (TXT) 139
4.6 The Aegis Secure Processor 140
4.7 The Bastion Architecture 142
4.8 Intel SGX 143
4.9 Sanctum 144
4.10 Ascend and Phantom 145

5 The Software Isolation Container (As Exemplified by Intel’s SGX) 147
5.1 SGX Physical Memory Organization 149
5.2 The Memory Layout of an SGX Enclave 153
5.3 The Life Cycle of an SGX Enclave 161
5.4 The Life Cycle of an SGX Thread 165
5.5 EPC Page Eviction 175
5.6 SGX Enclave Measurement 188
5.7 SGX Enclave Versioning Support 195
5.8 SGX Software Attestation 208
5.9 SGX Enclave Launch Control 220

6 Conclusion 230
Abstract

This manuscript is the first in a two part survey and analysis of the state of the art in secure processor systems, with a specific focus on remote software attestation and software isolation. This manuscript first examines the relevant concepts in computer architecture and cryptography, and then surveys attack vectors and existing processor systems claiming security for remote computation and/or software isolation. This work examines in detail the modern isolation container (enclave) primitive as a means to minimize trusted software given practical trusted hardware and reasonable performance overhead. Specifically, this work examines in detail the programming model and software design considerations of Intel’s Software Guard Extensions (SGX), as it is an available and documented enclave-capable system.

Part II of this work is a deep dive into the implementation and security evaluation of two modern enclave-capable secure processor systems: SGX and MIT’s Sanctum. The complex but insufficient threat model employed by SGX motivates Sanctum, which achieves stronger security guarantees under software attacks with an equivalent programming model.

This work advocates a principled, transparent, and well-scrutinized approach to secure system design, and argues that practical guarantees of privacy and integrity for remote computation are achievable at a reasonable design cost and performance overhead.

DOI: 10.1561/1000000051.
A user wishing to perform computation remotely faces a complex trade-off: how much trust can be placed in the remote system? How much of a performance overhead is considered acceptable for the given security properties? How strong an adversary can the remote system defend against? An ideal system would offer overhead-free trustworthy private remote computation with no assumptions of trust at all, yet no such system exists.

At one extreme, expensive cryptographic techniques including garbled circuits [Yao 1986] and fully homomorphic encryption [Gentry 2009] offer trust-free computation at prohibitive cost. A typical cloud computing scenario lies much closer to the opposite extreme: weak security guarantees achievable with minimal overhead assuming nearly unchecked trust in the remote system. This work aims to illustrate that significant security properties can be achieved given very modest trust in the remote system. A long lineage of secure processors explore the space of trusted hardware enabling inexpensive remote computation robust against a variety of threat models.

A rigorous conversation about security requires a precisely stated thread model: trusted hardware must be secure, meaning it must show
1.1. Secure Remote Computation

Secure remote computation (Figure 1.1) is the problem of executing software on a remote computer owned and maintained by an untrusted party, with some integrity and confidentiality guarantees. In the general setting, secure remote computation is an unsolved problem. Fully Homomorphic Encryption [Gentry, 2009] addresses the problem for a limited family of computations, but has an impractical performance overhead [Naehrig et al., 2011].

Intel’s Software Guard Extensions (SGX) is the latest iteration in a long line of trusted computing (Figure 1.2) designs, which aim to solve the secure remote computation problem by leveraging trusted...
Introduction

Figure 1.1: Secure remote computation. A user relies on a remote computer, owned by an untrusted party, to perform some computation on her data. The user has some assurance of the computation’s integrity and confidentiality.

hardware in the remote computer. The trusted hardware establishes a secure container, and the remote computation service user uploads the desired computation and data into the secure container. The trusted hardware protects the confidentiality and integrity of data while the computation is being performed on it.

SGX, Sanctum, and similar work rely on software attestation, like their predecessors, the TPM [TCG 2003] and TXT [Grawrock 2009]. Attestation (Figure 1.3) proves to a user that she is communicating with a specific piece of software running in a secure container hosted by the trusted hardware. The proof is a cryptographic signature that certifies the hash of the secure container’s contents. It follows that the remote computer’s owner can load any software in a secure container, but the remote computation service user is able to refuse to send private data to a secure container with a hash that does not match an expected value.

The remote computation service user verifies the attestation key used to produce the signature against an endorsement certificate cre-
1.1. Secure Remote Computation

Figure 1.2: Trusted computing. The user trusts the manufacturer of a piece of hardware in the remote computer, and entrusts her data to a secure container hosted by the secure hardware.

Figure 1.3: Software attestation proves to a remote computer that it is communicating with a specific secure container hosted by a trusted platform. The proof is an attestation signature produced by the platform’s secret attestation key. The signature covers the container’s initial state, a challenge nonce produced by the remote computer, and a message produced by the container.
ated by the trusted hardware’s manufacturer. The certificate states that the attestation key is only known to the trusted hardware, and only used for the purpose of attestation.

SGX stands out from its predecessors by the amount of code covered by the attestation, which is in the Trusted Computing Base (TCB) for the system using hardware protection. The attestations produced by the original TPM design covered the whole of the software running on a computer, and TXT attestations covered the code inside a VMX [Uhlig et al., 2005] virtual machine. In SGX, an enclave (secure container) only contains the private data in a computation, and the code that operates on it.

For example, a cloud service that performs image processing on confidential medical images could be implemented by having users upload encrypted images. The users would send the encryption keys to software running inside an enclave. The enclave would contain the code for decrypting images, the image processing algorithm, and the code for encrypting the results. The code that receives the uploaded encrypted images and stores them would be left outside the enclave. This example is illustrated in Figure 1.4.

![Image](image_url)

**Figure 1.4:** An example software application that uses SGX to implement a private function analyzing a medical image.

An SGX-enabled processor protects the integrity and confidentiality of the computation inside an enclave by isolating the enclave’s code
1.2. **SGX Lightning Tour**

and data from other software, including the operating system and hypervisor, and hardware devices attached to the system bus. At the same time, the SGX model remains compatible with the traditional software layering in the Intel architecture, where the OS kernel and hypervisor manage the computer’s resources.

This work discusses the original version of SGX, also referred to as SGX 1. While SGX 2 brings very useful improvements for enclave authors, it is a small incremental improvement, from a design and implementation standpoint. After understanding the principles behind SGX 1 and its security properties, the reader should be well equipped to face Intel’s reference documentation and learn about the changes brought by SGX 2 and more recent work.

1.2 **SGX Lightning Tour**

While this manuscript seeks to educate the reader of the challenges, history, and state of the art in secure processors for remote computation, this discussion is grounded in the example of Intel’s Software Guard Extensions (SGX), as it is an available, documented, and modern system that aims to offer useful security guarantees to remotely executed programs. This section presents a brief overview of the SGX platform, directing the reader to other sections of the manuscript for a deeper look at each aspect of SGX.

SGX sets aside a memory region, called the *Processor Reserved Memory* (PRM, §5.1). The CPU protects the PRM from all non-enclave memory accesses, including kernel, hypervisor and management engine (SMM, §2.3) accesses, and DMA accesses (§2.9.1) from peripherals.

The PRM holds the *Enclave Page Cache* (EPC, §5.1.1), which consists of 4 KB pages that store enclave code and data. The system software, which is untrusted, is in charge of assigning EPC pages to enclaves. The CPU tracks each EPC page’s state in the *Enclave Page Cache Metadata* (EPCM, §5.1.2), to ensure that each EPC page is assigned exclusively, belonging to exactly one enclave.
The initial code and data in an enclave is loaded by untrusted system software. During loading (§ 5.3), system software asks the CPU to copy data from unprotected memory (outside PRM) into EPC pages, and assigns the pages to the enclave being setup (§ 5.1.2). It follows that the initial enclave state is known to the system software.

After the enclave’s pages are loaded into EPC, the system software asks the CPU to mark the enclave as initialized (§ 5.3), at which point application software may execute code inside the enclave. After an enclave is initialized, the loading mechanism briefly described above is no longer available to system software.

While an enclave is loaded, its contents and configuration are cryptographically hashed by the CPU. When the enclave is initialized, this hash is finalized, and becomes the enclave’s measurement hash (§ 5.6). A remote party can communicate with the enclave to perform software attestation (§ 5.8) to convince itself that it is communicating with an enclave that has a specific measurement hash, and is running in a secure environment.

Execution flow can only enter an enclave via special CPU instructions (§ 5.4), similar to the mode switching mechanism for transitioning between user and kernel modes of execution in a typical system. An enclave must execute in protected mode, at ring 3, and uses virtual address translation as set up by the OS kernel and hypervisor.

To avoid leaking private information, a CPU executing enclave code does not directly service any interrupt, fault (e.g., a page fault) or VM exit. Instead, the CPU first performs an Asynchronous Enclave Exit (§ 5.4.3) to switch from enclave code to ring 3 code, and then services the interrupt, fault, or VM exit given scrubbed fault information. The CPU performs an AEX by saving the CPU state into a predefined area inside the enclave and transferring control to a predefined address outside of the enclave, replacing CPU registers with synthetic values.

The allocation of EPC pages to enclaves is delegated to the OS kernel (or hypervisor). The OS communicates its allocation decisions to the SGX platform via special ring 0 CPU instructions (§ 5.3). The OS can also evict EPC pages into untrusted DRAM and later load them back, again using dedicated CPU instructions. SGX uses a cryptographic
mechanism to enforce the confidentiality, integrity and freshness of the evicted EPC pages while they are stored in untrusted memory.

1.3 Outline

Reasoning about the security properties of Intel’s SGX requires a significant amount of background information that is currently scattered across many sources. For this reason, a significant portion of this work is dedicated to summarizing this prerequisite knowledge.

§2 summarizes the relevant subset of modern computer architecture and the micro-architectural properties of recent Intel processors. §3 outlines the landscape of trusted hardware systems, including cryptographic tools and relevant classes of attacks. Lastly, §4 briefly describes other trusted hardware systems as context in which SGX was created.

Following this background information, §5 provides a (sometimes painstakingly) detailed description of SGX’s programming model, largely drawing from Intel’s Software Development Manual.

A deep analysis of Intel’s enclave infrastructure is deferred to part II of this publication (§ II.2), and will analyze other public sources of information, such as Intel’s patents relevant to SGX, in order to fill in some of the missing detail in the SGX specification. This discussion is organized into an overview of Intel’s implementation of SGX (§ II.2.1), a discussion and analysis of the mechanism by which SGX offers memory access protection to an enclave (§ II.2.2 § II.2.3), and examines SGX as a system for remote attestation (§ II.2.5 § II.2.6). Finally, part II presents a security analysis of SGX overall, and discusses the classes of attacks against which SGX does not offer guarantees(§ II.2.7). The main focus of part II is a detailed review of SGX’s security properties to motivate and give context to the MIT Sanc- tum project (§ II.3) – a flexible, secure, and open source implementation of enclave-capable hardware that offers strong security guarantees against an insidious software adversary.
References


References


References


Loïc Duflot, Daniel Etiemble, and Olivier Grumelard. Using CPU system management mode to circumvent operating system security functions. CanSecWest/core06, 2006.


References


References


References


Minimal Intel® Architecture Boot Loader–Bare Bones Functionality Required for Booting an Intel® Architecture Platform. Intel Corporation, Jan 2010b. Reference no. 323246.


References


Intel® 64 and IA-32 Architectures Software Developer’s Manual. Intel Corporation, Sep 2015g. Reference no. 325462-056US.


References


References


Full text available at: http://dx.doi.org/10.1561/1000000051


References


References


References


References
