Secure Processors Part II:
Intel SGX Security
Analysis and MIT Sanctum
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
Foundations and Trends® in
Electronic Design Automation
Volume 11, Issue 3, 2017
Editorial Board

Editor-in-Chief

Radu Marculescu
Carnegie Mellon University
United States

Editors

Robert K. Brayton
UC Berkeley

Raul Camposano
Nimbic

K.T. Tim Cheng
UC Santa Barbara

Jason Cong
UCLA

Masahiro Fujita
University of Tokyo

Georges Gielen
KU Leuven

Tom Henzinger
Institute of Science and Technology Austria

Andrew Kahng
UC San Diego

Andreas Kuehlmann
Coverity

Sharad Malik
Princeton University

Ralph Otten
TU Eindhoven

Joel Phillips
Cadence Berkeley Labs

Jonathan Rose
University of Toronto

Rob Rutenbar
University of Illinois at Urbana-Champaign

Alberto Sangiovanni-Vincentelli
UC Berkeley

Leon Stok
IBM Research
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 II:
Intel SGX Security Analysis and MIT Sanctum 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 The Case for Hardware Isolation .................. 3
  1.2 Intel SGX is Not the Answer .................... 4
  1.3 MIT Sanctum Processor .......................... 5

2 An Analysis of Intel’s Software Guard Extensions (SGX) 7
  2.1 SGX Implementation Overview .................... 8
  2.2 SGX Memory Access Protection ................... 13
  2.3 SGX Security Check Correctness ................. 20
  2.4 Tracking TLB Flushes ............................ 28
  2.5 Enclave Signature Verification .................. 32
  2.6 Key Hierarchy and Derivation .................. 37
  2.7 SGX Security Properties ....................... 40

3 The MIT Sanctum Processor 58
  3.1 Threat Model ................................. 59
  3.2 Programming Model Overview ................... 61
  3.3 Protection Boundaries ......................... 67
  3.4 Security Primitives ............................ 67
  3.5 Hardware Modifications ....................... 69
  3.6 Software Design .............................. 76
Abstract

This manuscript is the second 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. The first part established the taxonomy and prerequisite concepts relevant to an examination of the state of the art in trusted remote computation: attested software isolation containers (enclaves). This second part extends Part I’s description of Intel’s Software Guard Extensions (SGX), an available and documented enclave-capable system, with a rigorous security analysis of SGX as a system for trusted remote computation. This part documents the authors’ concerns over the shortcomings of SGX as a secure system and introduces the MIT Sanctum processor developed by the authors: a system designed to offer stronger security guarantees, lend itself better to analysis and formal verification, and offer a more straightforward and complete threat model than the Intel system, all with an equivalent programming model.

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

Between the Snowden revelations and the seemingly unending series of high-profile hacks of the past few years, the public’s confidence in software systems has decreased considerably. At the same time, key initiatives such as cloud computing and the IoT (Internet of Things) are gaining popularity but require users to place much trust in the systems providing these services. We must therefore develop capabilities to build software systems with compelling security, and gain back our users’ trust.

This manuscript is the second in a two part survey of the state of the art in secure processor systems, with a specific focus on remote software attestation and software isolation. Part I [Costan et al., 2017] established relevant background in computer system design (§ 1.2) and security primitives (§ 1.3), and surveyed relevant prior work (§ 1.4). The same work discussed the attested software isolation container (enclave): a modern primitive for modular secure software and trusted remote computation, as exemplified by Intel’s Software Guard Extensions (§ 1.5).

This manuscript extends the discussion of enclaves and SGX by surveying the implementation and security properties of SGX (§ 2).
1.1. The Case for Hardware Isolation

and documents the authors’ concerns with its vulnerabilities to several classes of software attacks. Informed by the successes and shortcomings of SGX, this manuscript also discusses the MIT Sanctum processor (§ 3): a secure processor that offers an equivalent programming model with strong security guarantees against an insidious software threat model including cache timing and memory access pattern attacks. With this work, we hope to enable a shift in discourse in secure hardware architecture away from plugging specific security holes to a principled approach to eliminating attack surfaces.

1.1 The Case for Hardware Isolation

The best known practical method for securing a software system amounts to modularizing the system’s code in a way that minimizes code in the modules responsible for the system’s security. Formal verification techniques are then applied to these modules, which make up the system’s trusted codebase (TCB). The method assumes that software modules are isolated, so the TCB must also include the mechanism providing the isolation guarantees.

Today’s systems rely on an operating system kernel, or a hypervisor (such as Linux or Xen, respectively) for software isolation. However each of the last three years (2012-2014) witnessed over 100 new security vulnerabilities in Linux [cve, 2014a, Chen et al., 2011], and over 40 in Xen [cve, 2014b].

One may hope that formal verification methods can produce a secure kernel or hypervisor. Unfortunately, these codebases are far outside our verification capabilities: Linux and Xen have over 17 million [Anthony, 2014] and 150,000 [xen, 2015] lines of code, respectively. In stark contrast, the seL4 formal verification effort [Klein et al., 2009] spent 20 man-years to cover 9,000 lines of code.

Given Linux and Xen’s history of vulnerabilities and uncertain prospects for formal verification, a prudent system designer cannot include either in a TCB (trusted computing base), and must look elsewhere for a software isolation mechanism.
Fortunately, Intel’s Software Guard Extensions (SGX) \cite{McKeen2013,Anati2013} has brought attention to the alternative of providing software isolation primitives in the CPU’s hardware. This avenue is appealing because the CPU is an unavoidable TCB component, and processor manufacturers have strong economic incentives to build correct hardware.

### 1.2 Intel SGX is Not the Answer

Unfortunately, although the SGX design includes a vast array of defenses against a variety of software and physical attacks, it fails to offer meaningful software isolation guarantees. The SGX threat model protects against all direct attacks, but excludes “side-channel attacks”, even if they can be performed via software alone.

Alarmingly, cache timing attacks require only unprivileged software running on the victim’s host computer, and do not rely on any physical access to the machine. This is particularly concerning in a cloud computing scenario, where gaining software access to the victim’s computer only requires a credit card \cite{Ristenpart2009}, whereas physical access is harder, requiring trespass, coercion, or social engineering on the cloud provider’s employees.

Similarly, in many Internet of Things (IoT) scenarios, the processing units have some amount of physical security, but they run outdated software stacks that have known security vulnerabilities. For example, an attacker may exploit a vulnerability in an IoT lock’s Bluetooth stack and obtain software execution privileges, then mount a cache timing attack on its access-granting process, and obtain the cryptographic key that opens the lock.

Furthermore, the analysis of SGX documentation as described in Part I of this work reveals that it is impossible for anyone but Intel to reason about SGX’s security properties, because significant implementation details are not covered by the publicly available documentation. This is a concern, as the myriad of security vulnerabilities \cite{Wojtczuk2011,Rutkowski2009,Wojtczuk2009,Duflot2006,Rutkowski2008,Wojtczuk2008,Wojtczuk2008}...
1.3. MIT Sanctum Processor

In TXT [Wecherowski 2009, Embleton et al., 2010] in TXT [Grawrock 2009], Intel’s previous attempt at securing remote computation, show that securing the machinery underlying Intel’s processors is incredibly challenging, even in the presence of strong economic incentives.

If a successor to SGX claimed to protect against cache timing attacks, substantiating such a claim would require an analysis of its hardware and microcode, and ensuring that no implementation detail is vulnerable to cache timing attacks. Barring a highly unlikely shift to open-source hardware from Intel, such analysis will never happen.

A concrete example: the SGX documentation [Int 2013, 2014] does not state where SGX stores the EPCM (enclave page cache map). If the EPCM is stored in cacheable RAM, page translation verification is subject to cache timing attacks. Interestingly, this detail is unnecessary for analyzing the security of today’s SGX implementation, as we know that SGX uses the operating system’s page tables, and page translations are therefore vulnerable to cache timing attacks. The example does, however, demonstrate the fine nature of crucial details that are simply undocumented in today’s hardware security implementations.

In summary, while the principles behind SGX have great potential, the SGX design does not offer meaningful isolation guarantees, and the SGX implementation is not open enough for independent researchers to be able to analyze its security properties.

1.3 MIT Sanctum Processor

The Sanctum processor’s main contribution is a software isolation scheme that addresses the issues raised above: Sanctum’s isolation provably defends against known software side-channel attacks, including cache timing attacks and passive address translation attacks. Sanctum is a co-design that combines minimal and minimally invasive hardware modifications with a trusted software security monitor that is amenable to rigorous analysis and does not perform cryptographic operations using keys.

Sanctum achieves minimality by reusing and lightly modifying existing, well-understood mechanisms. For example, Sanctum’s per-
enclave page tables implementation uses the core’s existing page walking circuit, and requires very little extra logic. Sanctum is minimally invasive because it does not require modifying any major CPU building block. It only adds hardware to the interfaces between blocks, and does not modify any block’s input or output. The use of conventional building blocks limits the effort needed to validate a Sanctum implementation.

Sanctum demonstrates that memory access pattern attacks by malicious software can be foiled without incurring unreasonable overheads. Its hardware changes are small, small enough to present the added circuits, in their entirety, in Figures 3.9 and 3.10. Sanctum cores have the same clock speed as their insecure counterparts, as there are no modifications on the CPU core critical execution path. Using a straightforward page-coloring-based cache partitioning scheme with Sanctum adds a few percent of overhead in execution time, which is orders of magnitude lower than the overheads of the ORAM schemes [Goldreich, 1987; Stefanov et al., 2013] that are usually employed to conceal memory access patterns.

All layers of Sanctum’s TCB are open-sourced [MIT, 2017], and unencumbered by patents, trade secrets, or other similar intellectual property concerns that would disincentivize security researchers from analyzing it. The Sanctum prototype targets the Rocket Chip [Lee et al., 2014], an open-sourced implementation of the RISC-V [Waterman et al., 2014, 2015] instruction set architecture, which is an open standard. Sanctum’s software stack bears the MIT license.

To further encourage analysis, most of Sanctum’s security monitor is written in portable C++ which, once rigorously analyzed, can be used across different CPU implementations. Furthermore, even the non-portable assembly code can be reused across different implementations of the same architecture. In comparison, SGX’s microcode is CPU model-specific, so each micro-architectural revision would require a separate verification effort.
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


References


Full text available at: http://dx.doi.org/10.1561/1000000052