

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

IASM02/15
Tõnis Lusmägi 178217IASM

SGX - SECURITY & PERFORMANCE EVALUATION

Master's Thesis

Supervisor: Kolin Paul

PhD

Co-Supervisor: Peeter Ellervee

PhD

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutisüsteemide instituut

IASM02/15
Tõnis Lusmägi 178217IASM

SGX - TURVALISUSE JA JÕUDLUSE HINNANG

Magistritöö

Juhendaja: Kolin Paul

PhD

Kaasjuhendaja: Peeter Ellervee

PhD

Tallinn 2019

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Tõnis Lusmägi

2019-05-06

Abstract

Intel Software Guard Extensions (SGX) is collection of instruction set extensions and mechanisms for memory access that provide integrity and confidentiality guarantees on modern Intel processors. SGX is able to launch secure containers named enclaves that remain shielded from all CPU privilege levels.

This paper approaches SGX from security perspective, studying SGX from hardware to system level. Important security features like key hierarchy, encryption and attestation are broken down. Security evaluation is presented based on known vulnerabilities, attacks and exploits.

Performance of three SGX edge routine methods are benchmarked in two SGX environments - host and guest Linux, both fully capable of running SGX. This is achieved through experimental software offered by Intel. SGX virtualization uncovers interesting results both from systems and security perspectives.

This thesis is written in English and is 73 pages long, including 3 chapters, 28 figures, and 20 tables.

Annotatsioon

Intel Software Guard Extensions (SGX) on instruksiooni laienduste kogum ja mälu juurdepääsumehhanism, mis pakub andmete terviklikkuse ja konfidentsiaalsuse garantiisid modernsetel Inteli protsessoritel. SGX on võimeline käivitama turvalisi enklaavkonteinereid, mis on kaitstud kõikide CPU privileegitasemete eest.

See magistritöö läheneb SGX'le turvalisuse perspektiivist, uurides SGX'i riistvara tasemest kuni süsteemi tasemeni. Tähtsad turvafunktsioonid nagu võtmehierarhia, krüpteering ja atesteerimine seletatakse lahti. Turvalisuse hinnang esitatakse teadaolevate turvanõrkuste, rünnakute ja eksploatatsioonide analüüsil.

Kolmele erinevale SGX'i äärefunktsiooni meetodile tehakse jõudlustestid kahes erinevas SGX'i keskkonnas - *host* ja *guest* Linuxis, mõlemad võimelised täielikult rakendama SGX'i. See on saavutatud tänu Inteli eksperimentaalsele virtualiseerimistarkvarale. SGX'i virtualiseerimine paljastab huvitavad tulemused süsteemi kui ka turvalisuse perspektiividest.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 73 leheküljel, 3 peatükki, 28 joonist, 20 tabelit.

List of abbreviations and terms

AES-NI Advanced Encryption Standard New Instructions

AEX Asynchronous Enclave Exit

API Application Programming Interface

ASLR Address Space Layout Randomization

BIOS Basic Input Output System

CDF Cumulative Distribution Function

CPU Central Processing Unit

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

DDoS Distributed Denial-of-Service

DH Diffie-Hellman

DHKE Diffie-Hellman Key Exchange

DRAM Dynamic Random-Access Memory

ECall Enclave Call

EDL Enclave Definition Language

eFUSE Electronically Programmable Fuse

EPC Enclave Page Cache

EPID Enhanced Privacy ID

HVM Hardware Virtual Machine

IDL Interface Definition Language

IP Intellectual Property

IPP Integrated Performance Primitives

ISR Interrupt Service Routine

ISV Independent Software Vendor

ISVSVN Independent Software Vendor Security Version Number

KSS Key Sharing & Separation

KVM Kernel Virtual Machine

L1D Level 1 Data Cache

L1I Level 1 Instruction Cache

L1TF L1 Terminal Fault

LE Launch Enclave

LLC Last Level Cache

MAC Message Authentication Code

ME Management Engine

MFC Microsoft Foundation Class Library

MMU Memory Management Unit

NIST National Institute of Standards and Technology

NUC Next Unit of Computing

NVD National Vulnerability Database

OCall Outside Call

ORAM Oblivious Random Access Machine

OS Operating System

PCL Protected Code Loader

PMC Performance Monitoring Counters

PSW Platform Software

PvE Provisioning Enclave

QE Quoting Enclave

QEMU Quick Emulator

ROP Return-oriented Programming

SDK Software Development Kit

SDM Software Developer's Manual

SEH Structured Exception Handling

SGX Software Guard Extensions

SMM System Management Mode

SO Shared Object

SPS Server Platform Services

SSL Secure Sockets Layer

STL Standard Template Libraries

TCB Trusted Computing Base

TEE Trusted Execution Environment

TLS Transport Layer Security

TOCTTOU Time-of-Check-to-Time-of-Use

tRTS Trusted Run-Time System

TSX Transactional Synchronization Extensions

TXT Trusted Execution Technology

UEFI Unified Extensible Firmware Interface

uRTS Untrusted Run-Time System

VM Virtual Machine

VMM Virtual Machine Monitor

XEN Xenial

Contents

1	Introduction	14
2	Security Evaluation	16
2.1	SGX overview	16
2.1.1	Run-time system	18
2.1.2	Features at a glance	21
2.2	Key hierarchy & encryption schemes	22
2.3	Known vulnerabilities	27
2.4	Security research	29
2.4.1	Side-channel attacks	29
2.4.2	Side-channel mitigations	35
2.4.3	Speculative attacks	36
2.4.4	ROP attacks	36
2.5	Conclusion	37
3	System Setup	39
3.1	SGX support	39
3.2	Test hardware	41
3.3	Linux system	41
3.3.1	SGX driver	41
3.3.2	SGX SDK & PSW	42
3.4	Virtualized Linux system	43
3.4.1	KVM-SGX	44

3.4.2	QEMU-SGX	45
3.4.3	Guest system	47
3.5	Conclusion	49
4	Performance Evaluation	51
4.1	Testing methodology	53
4.2	Results	55
4.2.1	HotCalls	55
4.2.2	Switchless Calls	58
4.2.3	Comparison	60
4.3	Conclusion	63
5	Summary	65
	References	68
	Appendix 1 – SGX instructions	74

List of Figures

1	Protection rings	17
2	SGX hypothetical generic run-time system	18
3	Key hierarchy	23
4	SGX from hardware level to system level	39
5	Fetch SGX driver	42
6	Install the ISGX driver module	42
7	Fetch Linux-SGX	42
8	Build Linux-SGX	43
9	Install SGX PSW and SGX SDK	43
10	Working principles of KVM-SGX and QEMU-SGX	44
11	Fetch KVM-SGX	44
12	Configure and install KVM-SGX	45
13	Make KVM-SGX ramdisk	45
14	Fetch QEMU-SGX	46
15	Build and install QEMU-SGX	46
16	Check for, and load KVM modules	47
17	Check for, and load Virtio modules	47
18	Install and start Libvirt daemon	47
19	Create a virtual disk and boot a live image of Ubuntu	48
20	Launch QEMU-SGX	48
21	Install SGX PSW on Ubuntu	48
22	System 0: Arch Linux 5.0.0-mainline	49
23	System 1, Host: Arch Linux 5.0.0-kvm-sgx	50

24	System 2, Guest: Ubuntu 18.04.02 Linux 4.18.0-17-generic	50
25	Test 1, HotCalls, System 1: (a) Regular Calls, (b) HotCalls	57
26	Test 2, HotCalls, System 2: (a) Regular Calls, (b) HotCalls	57
27	Test 3, Switchless Calls, System 1: (a) Regular Calls, (b) Switchless Calls	59
28	Test 4, Switchless Calls, System 2: (a) Regular Calls, (b) Switchless Calls	59

List of Tables

1	Rules and limitations to SGX	20
2	Software level transformation key types [3]	24
3	Enclaves key policy fields [3]	24
4	Layout of KEYREQUEST data structure [3]	25
5	SGX SDK Integrated Performance Primitives (IPP) and Secure Sockets Layer (SSL) encryption schemes and primitives [5], [11], [12]	26
6	Enclave types by build configuration [5]	26
7	Enclave signing materials [5]	27
8	Status of known Intel SGX vulnerabilities	28
9	Microbenchmark latencies in HotCalls	56
10	Microbenchmark latencies in Switchless Calls	58
11	System 1 vs. System 2 RDTSCP overheads	60
12	System 1 vs. System 2 Regular Calls overheads	60
13	System 1 vs. System 2 HotCalls overheads	60
14	System 1 vs. System 2 Switchless Calls overheads	61
15	Regular ECalls vs. Hot ECalls	61
16	Regular OCalls vs. Hot OCalls	61
17	Regular ECalls vs. Switchless ECalls	62
18	Regular OCalls vs. Switchless OCalls	62
19	Hot ECalls vs. Switchless ECalls	62
20	Hot OCalls vs. Switchless OCalls	62

1 Introduction

Intel Software Guard Extensions (SGX) is a Trusted Execution Environment (TEE) for modern Intel processors. More specifically, SGX is a set of Central Processing Unit (CPU) instructions and mechanisms for encrypted memory access made available to the user through an extensive software stack [1]. SGX has a strong security model with multiple protection layers, offering security guarantees for confidentiality and integrity of data [1].

SGX technology, available since 6th generation Intel Core processors, is a change of direction for Intel, moving away from Trusted Execution Technology (TXT), the previous platform security technology [2]. SGX is considered quite new, although a second iterations of the technology, SGX2 and SGX Software Development Kit (SDK) v2.5 are currently available [3], [1]. There isn't a paradigm change in the new revision, additional features and enhanced security are introduced, while at it's core, still built on top of SGX1, which by it's own offers the full set of security guarantees [3].

SGX is suitable for cloud computing and offers additional security against malicious entities by cutting the cloud service provider out of the chain of trust [1]. Implementing SGX into cloud computing comes at a price. An additional virtualization layer to SGX is required, currently not natively available in SGX [3], [4]. Intel offers preliminary patches to support virtualization, while native instructions remain reserved for future CPUs [3], [4]. Virtualization of SGX remains a hot topic in computer engineering and cyber security.

SGX is supported on most modern Operating Systems (OS). Compatible SGX hardware is harder to find. Setting up SGX environments can be a complex procedure, especially for new adopters. The complexity rises exponentially when SGX virtualization is introduced, aimed for advanced users, still unambiguous.

SGX implies performance penalties for users. Research on SGX performance is scarce, but significant advances have been made. A recent addition to SGX software adds the ability to use a faster function call method [1]. While it is well tested and considered secure, wide-scale research into SGX function calls performance is lacking.

Topic of the thesis is SGX - Security and Performance Evaluation, presented in three chapters. The goals of this thesis are to analyze and assess SGX's security through hardware,

software and systems aspects. By evaluating SGX, a better understanding of relevant SGX vulnerabilities are obtained, which serves as an input to develop better protection mechanisms in future revisions of SGX, and may lead to development of new SGX applications, even integration of SGX into existing applications in need of better security. SGX performance evaluation gives a conclusive answer of the best calls method.

Problems to solve: SGX architecture and security research, SGX environments setup, programming microbenchmarks and testing with conclusive results.

Chapter 1 covers the full set of SGX's security features from hardware level to software level. Intel's documentation and publicly available information on vulnerabilities are used to assess SGX's security guarantees. Extensive literature review will uncover known SGX vulnerabilities. Chapter 2 focuses on system setup and covers all the bases to find compatible SGX hardware, and set up environments. Special focus is on SGX virtualization, which is in rapid development. Chapter 3 is performance evaluation, where SGX function calls performances are compared against each-other and against themselves across two SGX environments - guest and host Linux.

2 Security Evaluation

Intel SGX promises confidentiality and integrity for users data. To harness SGX's full potential, it should be understood first. This section looks at SGX's security features as stated by Intel. Through extensive literature review, SGX's real-world security properties are evaluated.

Evaluation starts from the CPU and works through the software stack. SGX is powered by a rich cryptosystem, which will be one of the main focus points of this paper. By understanding the cryptosystem and derivation of secure keys, potential vulnerabilities and attack vectors can be started to unravel. SGX intentionally doesn't protect against all types of attacks, the reasons for this are unknown, they are left for the user to solve.

This section presents a classification of known attacks on SGX, some of them disclosed publicly and mitigated, others, disregarded by Intel.

2.1 SGX overview

SGX is a set of instructions and mechanisms for memory access that provide integrity and confidentiality guarantees on Intel architecture processors [3]. Intel SGX encompasses three collections of instruction extensions, referred to as SGX1, SGX2 and OVERSUB [3]. SGX1 extensions enable an application to instantiate a protected container, named an enclave, which resides in an encrypted memory area (table 1) [3]. SGX2 extensions allow additional flexibility in runtime management of enclave resources and thread execution within an enclave (table 2) [3]. OVERSUB extensions enable Virtual Machine Monitors (VMMs) or other executives to manage the Enclave Page Cache (EPC) space more efficiently on the platform between virtualized entities (table 3) [3].

The virtual address space of the encrypted memory area - EPC has an upper limit of 128 MB and may vary depending on hardware [5]. EPC pages have 4 KB granularity [1]. Although there is no limit on the number of instantiated enclaves, Trusted Computing Base (TCB) size is still limited by the EPC size [5].

Intel 64 and IA-32 Architectures Software Developer's Manual (SDM) recognizes four privilege levels in IA-32 architecture's protection mechanism (figure 1) [3]. A greater level indicates less privilege [3]. Using protection levels improves the reliability of op-

erating systems, preventing programs or tasks operating at lesser privilege level from accessing a segment with a greater privilege [3]. Privilege levels can be interpreted as protection rings [3].

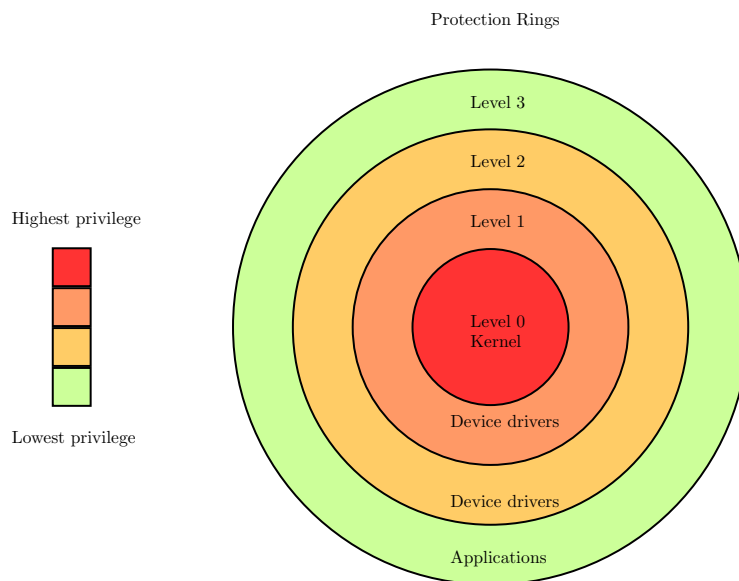


Figure 1. Protection rings.

Intel SGX is designed to protect against software attacks [5]. SGX enclave memory cannot be read or written from outside the enclave regardless of privilege level and CPU mode. This shields the enclave from the OS (ring 3 and ring 0), System Management Mode (SMM) and VMM, or another enclave [5]. Enclaves can be debugged with intel SGX debugger [5]. Enclaves cannot be entered via unauthorized function calls, jumps, register- or stack manipulation [5]. Only 32 or 64 bit CPU modes are supported to execute enclave code [5].

Protection against known hardware attacks include: enclave encryption using industry-standard encryption algorithms with replay protection, tapping the memory or connecting the Dynamic Random-Access Memory (DRAM) module to another system, regeneration of random memory encryption key after a power cycle and storing it into the CPU [5].

Other design considerations include side channel attacks or reverse engineering, from which Intel offers no security guarantees [5]. Protection against these attacks are left for

the developers to solve [5].

These software and hardware adversaries and countermeasures form the SGX security model [6].

2.1.1 Run-time system

SGX brings new programming constructs and principles best described as a hypothetical generic run-time system, consisting of following elements (figure 2) [1]:

- Untrusted Run-Time System (uRTS);
- Trusted Run-Time System (tRTS);
- Edge Routines;
- 3rd Party Libraries.

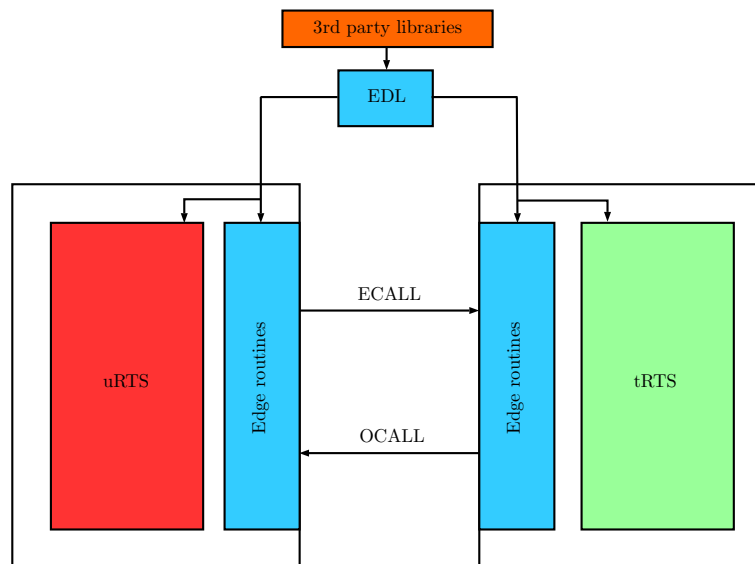


Figure 2. SGX hypothetical generic run-time system.

uRTS describes the code that executes outside of the enclave environment, and has functions like loading and managing enclaves, making calls to an enclave and receiving calls from within an enclave. tRTS describes the code that executes within an enclave, and

has functions like receiving calls into the enclave and making calls outside of an enclave, management of the enclave, and uses standard C/C++ libraries and run-time environment. The interface between the uRTS and the tRTS is formed by edge routines - Enclave Call (ECall) and Outside Call (OCall) functions that bind incoming and outgoing functions of the application with the enclaves (or vice versa). The Edger8r tool, supplied with SGX SDK, generates user specified edge routines automatically. The last element, 3rd party libraries describe any library that has been tailored to work inside an enclave [1].

SGX hypothetical generic run-time system implies rules and limitations to SGX (table 1) [5].

Enclave functions rely on special versions of the C/C++ runtime libraries, Standard Template Libraries (STL), synchronization and many other trusted libraries that are made available in the SGX SDK [5]. SGX allows to import 3rd party libraries into the enclave [5]. To be considered secure, libraries inside the enclave must follow rules that comply with internal enclave functions [5]. SGX developer reference manual includes a guide on enclave library porting and development [5].

Dynamic linking of enclave libraries is strictly prohibited by the enclave loader [5]. This restriction comes from the core security mechanism of SGX - enclave measurement [5]. This means that static linking remains the only option to include libraries inside the enclave, and furthermore, these libraries should not have any dependencies [5]. SGX SDK also includes trusted and untrusted simulation libraries, which don't require SGX hardware, but simulate SGX instructions instead [5].

Library functions inside the enclave which need to make ECalls and/or OCalls, need to be explicitly declared in proxy files named Enclave Definition Language (EDL) files [5]. These files are used by the Edger8r tool to create wrapper functions for enclave exports (used by ECalls) and imports (used by OCalls) [5]. Some trusted libraries come paired with their untrusted counterparts and EDL files [5].

Table 1. Rules and limitations to SGX.

Feature	Support	Comment
Languages	<input type="checkbox"/>	Native C/C++, interface functions limited to C
C/C++ calls to other Shared Objects (SOs)	<input checked="" type="checkbox"/>	Explicitly by OCalls
C/C++ calls to system provided C/C++/STL libraries	<input checked="" type="checkbox"/>	Only SGX trusted version of these libraries
OS Application Programming Interface (API) calls	<input checked="" type="checkbox"/>	Explicitly by OCalls
C++ frameworks	<input checked="" type="checkbox"/>	Microsoft Foundation Class Library (MFC), QT, Boost
Call C++ class methods	<input checked="" type="checkbox"/>	Including C++ classes, static and inline functions
Intrinsic functions	<input type="checkbox"/>	Only supported functions included in the SDK
Inline assembly	<input type="checkbox"/>	Same as previous
Template functions	<input type="checkbox"/>	Only supported in enclave internal functions
Ellipse (...)	<input type="checkbox"/>	Same as previous
Varargs (va_list)	<input type="checkbox"/>	Same as previous
Synchronization	<input type="checkbox"/>	Only SGX SDK spin-lock, mutex and condition variables
Threading support	<input type="checkbox"/>	Threads that run inside the enclave must be created within the untrusted application, support for thread synchronization within the enclave
Thread local storage	<input type="checkbox"/>	Only implicit via_thread
Dynamic memory allocation	<input checked="" type="checkbox"/>	Maximum heap size set at enclave creation
C++ exceptions	<input checked="" type="checkbox"/>	With performance impact
Structured Exception Handling (SEH) exceptions	<input checked="" type="checkbox"/>	SDK API allows to register exception handlers that handle limited set of hardware exceptions
Signals	<input checked="" type="checkbox"/>	Not supported inside the enclave

2.1.2 Features at a glance

Intel SGX can be described by its collection of unique security features. While the tools to develop secure enclaves are made readily available, SGX's strongest features are unlocked only in direct collaboration with Intel [5]. Features like attestation, provisioning and sealing form the outer-layer of SGX's security [5].

Attestation is the process of validating currently running software against its platform to be identical to their initially established identities [1]. Intel provides two attestation mechanisms with SGX: local- and remote attestation [1]. Successful attestation offers security guarantees for provisioning secrets into the secure enclave [1].

Local (intra-platform) attestation is a mechanism where software and platform identities are verified on the user's platforms (enclave to enclave) [1].

Remote (inter-platform) attestation is a mechanism where software and platform identities are verified remotely by a third party (Intel) [1], [7]. Remote attestation is performed by Intel's attestation service through an encrypted Transport Layer Security (TLS) connection between the attestation server and the user's SGX application [1]. The user is verified using a special Launch Enclave (LE) running on the user's platform, containing the remote attestation service license [5]. After a successful license check, the attestation server launches a special type of enclave in the user's application named Quoting Enclave (QE), that requests a report from the application's enclave [1]. QE authenticates the report and converts it into Enhanced Privacy ID (EPID) -signed quote which is delivered to the attestation server along with a manifest [1]. Attestation server uses EPID verification service to verify the quote's EPID signature [1]. The attestation server then compares the quote against a trusted configuration, verifying its identity [1]. Attestation server launches another special enclave in the user's application, named Provisioning Enclave (PvE), which is used to provision secrets into the user's enclave [1].

Sealing is an extended SGX feature, which enables to encrypt and store secrets to disk or the cloud [1], [8]. Sealing can be initiated locally, or remotely by the attestation service, that launches a seal enclave in the user's application, performing sealing on user's enclaves such as Intellectual Property (IP) enclaves (Protected Code Loader (PCL)) [1], [5]. Sealing enables yet another SGX feature - the ability for enclaves to survive power transitions [1]. Every enclave generates a token (EINITTOKEN) for the SGX application during build time [5]. This token is used to verify that an enclave is permitted to launch,

and secures that secrets in the enclave, remain accessible only to the enclave that created it [5].

Other extended SGX features are Switchless Calls and Key Sharing & Separation (KSS) [5]. The first is a fast ECall/OCall method and the latter is part of key derivation, adding additional control over the key derivation process [5].

Lastly, SGX uses a new programming language named EDL based on Interface Definition Language (IDL) family, and acts as a static proxy between the trusted and untrusted SGX constructs [5]. This includes linking the enclave with trusted headers and libraries, and defining trusted functions that can enter and exit enclaves [5].

2.2 Key hierarchy & encryption schemes

SGX provides software with access to keys unique to each CPU that are rooted into the package during manufacturing [3]. While largely undocumented, it is known that these keys are the **root provisioning key** and the **root seal key** [9] - embedded into the processor's internal hardware registers (CREGs) - Electronically Programmable Fuses (eFUSES), named CR_SEAL_FUSES, which are 128 bits and part of the CPU package [3], [10].

The **root provisioning key** is randomly generated by a special purpose offline key generation facility, delivered to Intel via a secure factory network [9]. This key proves to the TCB that it's a genuine SGX CPU [9]. The **root seal key** is generated and programmed into the eFUSE during the manufacturing of the processor [9]. While the root provisioning key is known and retained by Intel, the root seal key is not - it's generated by an internal automated process and is only known to the specific CPU itself [9]. All keys except the root provisioning key include the root seal key in their derivations, rendering them unknown to Intel [9]. The root provisioning key, the root seal key and Independent Software Vendor Security Version Number (ISVSVN) together form the hardware TCB, which is the first key transformation that cascades into subsequent key derivations (figure 3) [9].

Each enclave requests keys using the EGETKEY leaf function. This key is based on enclave parameters such as measurement, the enclave signing key, security attributes of the enclave and the hardware TCB key [3]. This second key transformation happens on the software level [9]. By using a key derivation mechanism based on enclave properties,

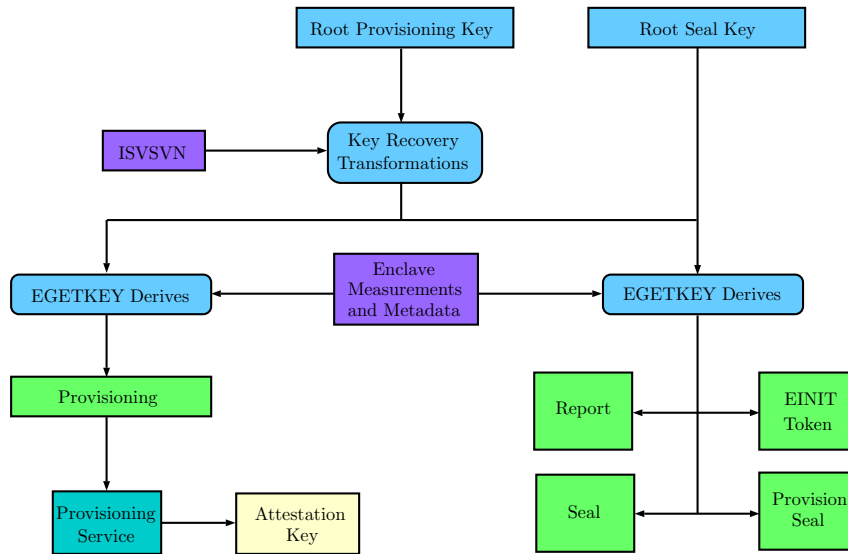


Figure 3. Key hierarchy.

it is guaranteed that each enclave requesting a key gets a unique key, only accessible by the respective enclave [3]. It also guarantees that an enclave gets the same key at different request [3].

Table 2 lists the key types that can be generated with EGETKEY leaf function, and table 3 lists the enclave key policy fields, which become the basis for key derivation [3], [5], [9]. EGETKEY combines the values of these two tables using a pseudo random function that populates the KEYREQUEST data structure and generates the derived key (table 4) [9]. Enclave keys can be derived from common key policy fields, e.g., when multiple enclaves request keys based on common signing identity (MRSIGNER), then that group of enclaves can use the same key [3]. Key sharing and derivation in SGX is handled by Diffie-Hellman Key Exchange (DHKE) [5].

Table 2. Software level transformation key types [3].

Key name	Description
EINITTOKEN_KEY	Key used to calculate the Message Authentication Code (MAC) on EINITTOKENs. Tokens are verified in the EINIT leaf function
PROVISION_KEY	Key used by attestation provisioning software to prove to the remote party that the CPU is genuine and identify the current executing TCB
PROVISION_SEAL_KEY	Same as previous, but key is used for sealing data to disk
REPORT_KEY	Used to calculate the MAC on the REPORT structure. EREPORT leaf function calculates the MAC and destination enclave uses the report key to verify the MAC
SEAL_KEY	General purpose key for the enclave, used to encrypt and calculate MAC of secrets on disk. There are two types of seal keys. One is based on MRENCLAVE, other on MRSIGNER.

Table 3. Enclaves key policy fields [3].

Field	Description
MRENCLAVE	SHA-2 hash measurement of the enclave calculated at build (enclave identity)
MRSIGNER	SHA-2 hash of the public key used to sign the enclave's SIGStruct (signing identity)
NOISVPRODID	Independent Software Vendor (ISV) Product ID assigned by enclave signer through the SIGStruct. This field indicates that Product ID will not be used
CONFIGID	Enclave configuration ID
ISVFAMILYID	ISV assigned Family ID
ISVEXTPRODID	ISV extended Product ID
RESERVED	Must be zero

Table 4. Layout of KEYREQUEST data structure [3].

Field	Description
KEYNAME	Key type
KEYPOLICY	Key policy input used in key derivation
ISVSVN	ISVSVN used in key derivation
RESERVED	Must be zero
CPUSVN	CPU's SVN used in key derivation
ATTRIBUTEMASK	Mask to define ATTRIBUTES bits used in key derivation
KEYID	Key wear-out protection
MISCMASK	Mask to define MISCSELET bits used in key derivation
CONFIGSVN	Enclave configuration's SVN used in key derivation
RESERVED	Must be zero

SGX SDK provides an enclave signing tool named *sgx_sign*, which is used to sign the enclaves using cryptography primitives provided in the SGX SDK crypto libraries (table 5) [5]. SGX signing tool is able to sign and launch all enclaves using a 1-step signing process, except production enclaves (table 6) [5]. Production enclaves must follow a 2-step signing process [5]. First step uses *sgx_sign* tool at the end of the enclave build process, where it generates the enclave signing materials (table 7) [5]. ISV takes the signing materials to an external signing platform/facility retaining a private key, which is used for signing, then taken back to the build platform [5]. At the second step, ISV runs the *sgx_sign* tool with a special command, that populates the enclave's metadata section with a public key hash and a signature [5]. Enclave's metadata is stored in dedicated EPC pages [2]. These pages are not mapped to any of the enclave's address spaces, but mapped separately to be used by the SGX software [2]. SGX can furthermore harness the CPU's Advanced Encryption Standard New Instructions (AES-NI) [5].

SGX attestation uses Intel EPID, which is an anonymous group signature scheme for remote authentication and secret provisioning [1], [7]. EPID allows platforms to cryptographically sign objects while preserving the signers privacy [1]. In EPID, each signer has a unique private key, while verifiers use a common public key (EPID Group ID key) to verify individual signatures. This ensures user privacy, as they cannot be uniquely identified [1]. EPID never exposes the CPU outside of the platform [1].

Table 5. SGX SDK IPP and SSL encryption schemes and primitives [5], [11], [12].

Name	Usecases
AES-CTR	Encryption and decryption of input data stream. CTR primitive increments a counter on successful calls, increment block is used on subsequent operations on the same data
AES-GCM	Encryption and decryption of data, utilizes AES-NI. PCL
CMAC	Hash calculation of dataset
ECC	Shared key generation for two participants of the cryptosystem
ECC-ECDSA	Encryption and decryption of data, key exchange
HMAC	Hash calculation of input data buffer
RSA-PKCS #1 v1.5	Signing enclave material
SHA-1	Hash calculation of input data buffer
SHA-2	Hash calculation of input data buffer, PCL

Table 6. Enclave types by build configuration [5].

Type	Description
Simulation	Simulated enclave does not need an SGX processor
Debug	Intel SGX debugger can connect to debug enclaves. Debug flag can be set in parallel with all enclave types
Prerelease	Prerelease enclave is identical to production enclave, but requires only 1-step signing by <i>sgx_sign</i> tool
Release	Production enclave requires 2-step signing, the latter, with a white-listed key by Intel

EPID signature scheme allows to overcome well-known privacy concern of standard asymmetric cryptographic signature algorithms, where a key is associated with specific hardware performing the quoting operation [1]. Usually a small number of keys are used during the life-cycle of the platform, that make it possible for third parties to collude and track users [1].

Intel PCL is intended to protect the confidentiality of the enclave code [5]. SGX guarantees integrity of code and confidentiality and integrity of data at run-time [5]. To provide confidentiality of offline binaries, PCL is used, that symmetrically encrypts the enclave shared object at build time and decrypts at enclave load time [5].

Table 7. Enclave signing materials [5].

File	Format	Description
Enclave file	Shared object	Standard shared object
Signed enclave file	Shared object	Enclave file with a signature, signed by <i>sgx_sign</i>
Configuration file	XML	Enclave configuration file
Key file	PEM	Contains unencrypted RSA 3072-bit key with public exponent of 3
Enclave hex file	RAW	Dump of all enclave signing material to be signed with the private RSA key
Signature file	RAW	Dump of the signature generated by ISV's external signing facility using RSA-PKCS #1 v1.5 and SHA-2 hash
Metadata file	RAW	Dump of SIGStruct metadata of the signed enclave. This file is used by Intel to to whitelist production enclaves

PCL encryption/decryption is handled by Intel remote attestation service [5]. The sealed enclave becomes the IP enclave [5]. While PCL does not encrypt end-to-end, it encrypts the relevant IP sections [5]. The sections remaining unencrypted are detailed in SGX developers reference manual, and should be taken into consideration by the developers [5]. PCL uses modified code snippets from OpenSSL1.1.0g library, which become part of the IP enclave's TCB [5].

2.3 Known vulnerabilities

National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD) lists following Common Vulnerabilities and Exposures (CVE) in Intel SGX [13], [14]:

- **CVE-2019-0122** Double free in Intel(R) SGX SDK for Linux before version 2.2 and Intel(R) SGX SDK for Windows before version 2.1 may allow an authenticated user to potentially enable information disclosure or denial of service via local access.
- **CVE-2018-18098** Improper file verification in install routine for Intel(R) SGX SDK

and Platform Software for Windows before 2.2.100 may allow an escalation of privilege via local access.

- **CVE-2018-3615** Systems with microprocessors utilizing speculative execution and Intel software guard extensions (Intel SGX) may allow unauthorized disclosure of information residing in the L1 data cache from an enclave to an attacker with local user access via a side-channel analysis.
- **CVE-2018-3689** AESM daemon in Intel Software Guard Extensions Platform Software Component for Linux before 2.1.102 can effectively be disabled by a local attacker creating a denial of services like remote attestation provided by the AESM.
- **CVE-2018-3626** Edger8r tool in the Intel SGX SDK before version 2.1.2 (Linux) and 1.9.6 (Windows) may generate code that is susceptible to a side channel potentially allowing a local user to access unauthorized information.
- **CVE-2017-5691** Incorrect check in Intel processors from 6th and 7th Generation Intel Core Processor Families, Intel Xeon E3-1500M v5 and v6 Product Families, and Intel Xeon E3-1200 v5 and v6 Product Families allows compromised system firmware to impact SGX security via incorrect early system state.
- **CVE-2017-5736** An elevation of privilege in Intel Software Guard Extensions Platform Software Component before 1.9.105.42329 allows a local attacker to execute arbitrary code as administrator.

Table 8 pairs CVEs with Intel IDs and lists the status of each vulnerability along with a Common Vulnerability Scoring System (CVSS) V3 score [15].

Table 8. Status of known Intel SGX vulnerabilities.

CVE	Intel ID	CVSS V3	Status
2019-0122	SA-00217	7.1	Mitigated
2018-18098	SA-00203	7.3	Mitigated
2018-3615	SA-00161	6.4	Mitigated
2018-3689	OSS-10004	5.5	Mitigated
2018-3626	SA-00117	4.7	Mitigated
2017-5691	SA-00076	9.0	Mitigated
2017-5736	SA-00117	8.8	Mitigated

Intel, and nearly the entire technology industry, follows a practice called Coordinated Disclosure [16], under which a cybersecurity vulnerability is generally publicly disclosed only after mitigations are deployed [17].

Majority of CVEs are mitigated with SGX SDK and SGX Platform Software (PSW) updates [18], [19], [20], [17]. CVE-2018-3615 is caused by speculative execution side-channel method called L1 Terminal Fault (L1TF), similar vulnerability to Spectre/Meltdown, mitigated by a set of firmware, OS and microcode updates [21], [22], [23], [24]. CVE-2017-5691, rated *critical*, identified a problem in Intel Server Systems, Next Unit of Computing (NUC), and Compute Stick firmwares, mitigated by firmware updates for affected products [25].

2.4 Security research

This section covers all known and researched SGX vulnerabilities and mechanism to break the confidentiality and integrity of enclaves [26]. Mitigations and workarounds are also discussed. Vulnerabilities are categorized and assessed.

2.4.1 Side-channel attacks

AsyncShock: Exploiting Synchronization Bugs in Intel SGX Enclaves research paper discusses an attack model, where the adversary has full control over the SGX environment, full control of the OS and code prior to transferring control over to the enclave. Adversary can interrupt and resume SGX threads, which is the main attack vector exploited in the paper. Adversary's goal is to compromise the confidentiality and integrity of the SGX enclave. Crashing the thread is not part of the attack [27].

AsyncShock is a tool for exploiting synchronization bugs of multithreaded code running in the enclave by manipulating thread scheduling used to execute an enclave. Atomicity bugs exploited are Use-After-Free and Time-of-Check-to-Time-of-Use (TOCTTOU) [27].

Use-After-Free can be considered a parent to double free bug [28], which is mitigated in the SGX SDK (CVE-2019-0122) [18].

TOCTTOU bug delays the writer thread to interrupt enclave execution after the check,

and then letting the writer thread to proceed. This timing-based attack window opens after delaying the writer thread for empirically determined time between check and use. This allows to extract the data buffer from within the enclave [27].

Only protection against TOCTTOU is prohibiting threading or decoupling of enclave and user threads. Nevertheless, AsyncShock can still be used to force an Asynchronous Enclave Exit (AEX) [27].

Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems research paper introduces controlled-channel attacks able to extract complete text documents, and outline JPEG images from enclaves. Researchers attack SGX through Haven and uncover a page-fault vulnerability applicable to all SGX enclaves [29].

SGX leaves all page tables under the control of the host OS, and implements an independent memory protection mechanism. Memory access will fail if disallowed under either of the mechanisms. Researchers' attack code enables access to page tables. By editing page tables accordingly, the code inside the enclave will cause a page-fault [29].

Page-fault causes SGX to relinquish control over page-fault handler to the OS. SGX reveals the page number of the faulting address to the OS, but not the offset within the page. The OS is called to map and unmap these memory regions in the enclave and change their page-access permissions. This uncovers the map of the enclaves memory layout with loading addresses and -times of binaries. That information is used to trigger the attack on an enclave [29].

Researchers attach a malicious page-fault handler to an enclave to reliably extract data from within. Text documents and JPEG images are successfully extracted [29].

Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution research paper discusses a page table-based attack without forcing a page-fault. Page faults are already known to be a major vulnerability in SGX, allowing adversaries to extract enclave secrets. While this is true, multiple state-of-the-art defenses have been deployed to mitigate them [30].

However, it is shown that page table-based attacks go beyond page faults, and it is possible to observe enclaved page accesses by exploiting other side-effects of the address translation process. Two attack vectors are uncovered, first infers enclaved memory ac-

cesses from page table attributes, second attack vector does the same thing through the cache. This allows to recover EdDSA session keys from the SGX cryptography library [30].

Attack model assumes a standard SGX threat model, where the adversary has full control over the OS's thread scheduler in a classic Memory Management Unit (MMU) based system architecture that hides enclave page faults from system software, and OS controls enclave page mappings [30].

The adversary infers page access patterns from an enclaved execution that never suffers a page fault. Adversary extracts the information from an enclave with a single run that remains undetected to Intel's remote attestation service. This attack was implemented as an extension to Graphene-SGX and uncovered two vulnerabilities in SGX Edger8r tool [30].

Intel publicly disclosed these vulnerabilities, and released updates to SGX SDK and SGX PSW, mitigating the vulnerabilities (CVE-2018-3626, CVE-2017-5736) [17].

Stacco: Differentially Analyzing Side-Channel Traces for Detecting SSL/TLS Vulnerabilities in Secure Enclaves research paper discusses control-flow inference attacks against the trusted SGX SSL library responsible for TLS. This man-in-the-kernel attack collects execution traces of enclave programs at the page level, cacheline level or branch level while acting as a middleman between the trusted and untrusted parts of the SGX program. The entity in the middle is a differential analysis framework named STACCO, that dynamically analyses the SSL/TLS implementation and detects vulnerabilities used to exploit the SSL library using a Bleichenbacher attack, which targets RSA-PKCS #1 v1.5 [31].

STACCO, implemented in Graphene-SGX, was used to successfully break the SSL library's *PreMasterSecret* encrypted by a 4096-bit RSA public key. While it took close to 60 000 queries, other libraries like GnuTLS and mbedTLS-SGX took roughly 50 000 and 25 000 queries respectively to break a single block of AES ciphertext [31].

STACCO authors suggest countermeasures in three layers of SGX. First countermeasure is prevention of control-flow inference attacks in page-level, cache-level and branch-level. Second countermeasure is to patch vulnerabilities in SSL/TLS implementations, specifically, eliminate the error call function from RSA-PKCS #1 v1.5 padding check and de-

ryption functions. Third countermeasure is elimination of the root cause of Bleichenbacher attack, the RSA-based key exchange from the SSL implementation, moving to Diffie-Hellman (DH) instead [31].

CacheZoom: How SGX Amplifies The Power of Cache Attacks research paper proposes an attack tool named CacheZoom, which is able to track all memory access of SGX enclaves. Authors hypothesize that Intel's disregard for side-channel attacks in SGX amplifies cache-based attacks. Researchers demonstrate how CacheZoom is able to recover AES keys used for enclave data encryption, completely compromising SGX's security [32].

CacheZoom creates a high-resolution side channel in SGX by monitoring L1 data and instruction caches (or Last Level Cache (LLC)) and retrieving maximum leakage using a Prime+Probe attack. This forces an interrupt in enclave's execution and identifies enclave's memory accesses. L1 cache is virtually addressed, by knowing the enclave's page boundary, the accessed set is revealed [32].

High-Resolution Side Channels for Untrusted Operating Systems research paper presents two side channels in the untrusted OS. These attacks are solely based on the page-fault channel. High-precision timer and cache misses are used to open up spacial and temporal high-resolution channels. The OS can break into the application before and after an enclaved memory access [33].

Researchers were able to break into the enclaved MapReduce application and Libjpeg library, successfully retrieving text documents and images [33].

Authors suggest mitigations by partitioning caches in a way that malicious programs cannot access the sensitive parts. Another defense mechanism is introduction of noise that conceals cache misses and obfuscates it's cause, effectively masking a real cache miss as a random memory access. These mitigations are not widely used because of high cost and availability of cryptography. Demonstrated side channels highlight that a broader codebase might be the subject of attacks when the OS is the adversary. Authors discuss a mitigation technique called T-SGX which disables side channels based on page-faults and interrupts, but adds a noticeable overhead [33].

Cache Attacks on Intel SGX research paper discloses enclave vulnerabilities against cache-timing attacks. A case study is presented using an access-driven cache-timing at-

tack on AES running inside an enclave. Researchers practically demonstrate that an AES key can be retrieve in less than 10 seconds using Neve and Seifert’s elimination method together with a cache probing mechanism powered by Intel Performance Monitoring Counters (PMC) [34].

The attack is launched at root-level, from which SGX is specifically designed to protect against. The case study targets an older version of OpenSSL with Gladman AES implementation, which is not used in the SGX SDK [34].

Current version of SGX SDK 2.5 uses AES-GCM, which protects enclaves from software-based side channel attacks [5].

Malware Guard Extension: Using SGX to Conceal Cache Attacks research paper demonstrates a software-based side channel attack, where a malicious code runs inside an enclave and targets co-locating enclaves of the same hardware. This malware is designed to break through the hypervisor into other tenants’ enclaves. Researchers demonstrate the attack in a native environment and also across multiple Docker containers. Prime+Probe cache side channel is used to break through the SGX enclaves to extract 96% of an RSA private key, leaving a single trace, and the full key leaving 11 traces within 5 minutes. The targeted RSA implementation uses a constant-time multiplication primitive (mbedtls) [35].

The attack further exploits the DRAM timing difference, and uses the DRAM mapping function to map physical address bits. DRAM is alternately accessed at two virtual addresses of the same bank, but different rows. Memory is scanned sequentially to cause a row conflict. In doing so, exact cache set for the two virtual addresses can be obtained [35].

Researchers suggest countermeasures such as eliminating timers, adding the cloud provider to the root of trust, and an additional secure memory module. Researchers discuss elimination of timers as not a viable countermeasure, because a high-resolution Prime+Probe attack can be mounted without them. The discussion continues with adding cloud providers to the root of trust. While it enables the cloud provider to scan enclaves for malware, this defeats the purpose of SGX. Researchers suggest a signature-based virus scanner for enclaves, but the methods how to securely implement the feature are not detailed. The most viable countermeasure of an additional secure memory module can be easily implemented into the SGX driver [35].

Researchers conclude that SGX is ideal to conceal malware, as SGX shows no enclave activity in system's performance counters. This core security feature prevents SGX software-based side channel attacks, also hides malicious code in a similar manner, making it undetectable [35].

Software Grand Exposure: SGX Cache Attacks Are Practical research paper focuses on uncovering more side channel attacks on SGX. Page-fault is recognized as the most dangerous SGX side channel method, but raises the question of other cache vulnerabilities such as cache access pattern monitoring. The goal of the research is to design a practical cache attack that is hard to mitigate using known defenses. To succeed, researchers need to keep enclave execution uninterrupted and implement a low-noise cache monitoring technique. This method is able to efficiently extract the entire RSA-2048 key during RSA decryption along with data from within the enclave [36].

Sophisticated Prime+Probe attack together with enclave isolation is used to reduce cache noise and extract enclave's secrets [36].

Noise isolation techniques are used such as self-pollution that leverages the L1 cache structure which is divided into Level 1 Data Cache (L1D) and Level 1 Instruction Cache (L1I) and never maps to cache lines of interest for the attacker [36].

Uninterrupted execution of enclaves prevents AEXs, which normally invoke the OS's Interrupt Service Routine (ISR) and induces noise into cache line. By making the enclave execution uninterrupted, the enclave remains unaware of the attack. This is achieved by rerouting the interrupt controller's data-path such, that interrupts never reach the attack core the enclave is isolated to. Only exception to this is the dedicated CPU core timer, that can only handle interrupts on the specific core itself. This is overcome by reducing the interrupt timers frequency to 100Hz. This gives a 10ms window during which an undetected attack can be launched [36].

To further reduce noise, clever cache monitoring is used. Prime+Probe determines cache evictions by measuring the time required for accessing memory that maps to cache lines. Successful cache misses and hits can be determined, but reading the data from the L1 and L2 caches introduces noise. Researchers use Intel PMC to monitor cache evictions. However, Prime+Probe code is still used to prime the cache lines. The victim evicts the cache, resulting in a cache miss. PMC identifies the misses and determines the victim's cache lines [36].

PMC is an expensive operation in the Prime+Probe cycle. To detect all cache misses and learn the victims complete cache access pattern, PMC must execute at a high frequency [36].

Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing research paper explores a new, critical side channel attack against SGX, named - branch shadowing attack, which reveals fine-grained control flows (branches) of an enclave. This attack exploits SGX's mechanisms, where after a secure context switch, branches are not cleared, leaving fine-grained traces into the branch-prediction side channel [37].

While this exploit is not straightforward, researchers develop novel exploitation techniques which even break Oblivious Random Access Machine (ORAM) schemes, Sanctum, SGX-Shield, and T-SGX - the state-of-the-art side-channel mitigations [37].

2.4.2 Side-channel mitigations

T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs research paper proposes a mitigation solution to controlled-channel attacks. T-SGX leverages a commodity component of modern Intel CPU's - Transactional Synchronization Extensions (TSX), which implements a restricted hardware transactional memory that forces an abort in an ongoing transaction when unusual events are detected. TSX abort suppresses the notification of errors to the OS. This can be used to eradicate all known controlled-channel attacks, including the page-fault attack [38].

Researchers implemented T-SGX as compiler-level scheme that automatically transforms the enclave into a secure one. T-SGX's security properties are evaluated and demonstrated that it indeed mitigates all known controlled-channel attacks. While it is secure, it also promises less overhead than other state-of-the-art mitigations [38].

Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory research paper proposes another TSX-based side channel protection named Cloak. Cloak promises protection against all known cache-based side channel attacks with low performance overhead. Vulnerable code can be retrofitted with Cloak and it can be applied to code within an enclave, blocking all side channels [39].

2.4.3 Speculative attacks

FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution research paper uncovered a speculative execution bug in modern Intel CPU's, which is exploited to leak plaintext enclave secrets from the L1 cache. Researchers completely break SGX, by extracting full cryptographic keys from Intel's architectural enclaves and validate them by launching rogue production enclaves and forging arbitrary attestation responses. Researchers use Foreshadow to extract remote attestation keys, compromising millions of devices [22].

Intel acknowledged Foreshadow and disclosed it as L1 Terminal Fault (L1TF), a speculative execution side channel cache timing vulnerability [24], [22]. L1TF is a hardware/architecture-level vulnerability in the CPU associated with infamous Spectre and Meltdown bugs [22], [24], [40], [41]. Foreshadow was mitigated in CVE-2018-3615 [21].

Researchers also discuss a second version of Foreshadow (Next Generation), which affects Virtual Machines (VMs), VMMs, OS kernel memory, and SMM memory, also mitigated in CVE-2018-3615 [23], [21].

2.4.4 ROP attacks

Hacking in Darkness: Return-oriented Programming (ROP) against Secure Enclaves exploits memory corruption vulnerabilities in enclave code, exploring new exploitation techniques, different from known vulnerabilities [42].

Researchers develop Dark-ROP, which exploits memory corruptions in the enclave software. Dark-ROP differs significantly from traditional ROP attacks because of SGX's secure hardware-level implementation [42].

Dark-ROP cleverly overcomes SGX's security techniques and completely breaks the enclave's memory protections, tricking SGX hardware into leaking enclave secrets, defeating even remote attestation [42].

Dark-ROP relies on page-faults and demonstrates how to break into enclaves using AEXs [42].

Practical Enclave Malware with Intel SGX researchers disclose possibilities to stealthily place malware inside an enclave, that impersonates its host application. This type of concealed malware can steal or encrypt data for extortion, send phishing emails or mount Distributed Denial-of-Service (DDoS) attacks [43].

SGX-ROP uses TSX primitives to construct a code-reuse attack from within an enclave - executed by the host application. SGX-ROP can bypass Address Space Layout Randomization (ASLR), stack canaries, and address sanitizers. Researchers demonstrate that SGX's security threats outweigh its protections [43].

2.5 Conclusion

Intel SGX's security stems from the physical hardware. Intel programs two keys into the CPU's hardware registers, the **root provisioning key** known to Intel, and the **root seal key** known only to the CPU. This alleviates privacy concerns in using SGX, as the key derivation process transforms the hardware keys along with a serial number to form the hardware TCB. The root seal key is present in all subsequent key derivations, making them unknown to Intel.

SGX uses modern encryption schemes and primitives, mostly considered secure. Additional layer of security and acceleration is given by Intel AES-NI. RSA and AES are known to be vulnerable to exploitations. This raises the question of the useful life-cycle length of encryption primitives. With advances in quantum computing promising to break modern encryption, new techniques and algorithms must be considered, such as AI-based, blockchain-based and quantum cryptography [44].

Research on known SGX vulnerabilities offers some insight into the future of SGX. Intel SGX does not protect against all side channels, acting as an open invitation to researchers and adversaries.

Attacks on SGX fall into three categories: side channel, speculative and ROP. Usually, a combination of attacks are used. Researchers believe that Intel's disregard for side channel attacks amplify them, which is shown to be true. Literature review uncovered how software bugs, SGX's intended mechanisms, cache architecture, DRAM and speculation are used to open side-channels, and completely breaks SGX - steal data and keys - even from Intel's remote attestation service, compromising millions of devices.

SGX vulnerabilities stemming from bugs in the software stack, and from speculation, are mitigated by Intel. The most devastating attacks exploit SGX's intended mechanisms, such as AEX, which is used to force page faults, now become the main vulnerability in SGX.

Mitigation techniques and additional security for side channels are offered through third parties - researchers trying to develop novel, secure, and easy to use methods - securing users against SGX vulnerabilities that Intel doesn't.

TSX offers protections against side channels, able to close them from within an enclave. Others use TSX to put malicious code inside an enclave, completely concealing it from the platform and anti-virus software. Changes in Intel's CPU architecture are needed, which must offer better security guarantees not only for SGX, but for the whole platform.

ROP is expected to be the next major SGX attack vector, focusing on alternatives to side-channel methods.

3 System Setup

SGX setup can be straightforward or complex. The user has to keep in mind that not all hardware is suitable for SGX.

This section details how to set up SGX environments from native to virtualized platforms. This includes SGX software installation on an unofficial Linux distribution, and setting up Intel’s experimental software. While SGX comes with an installation manual, there are gaps in-between. This section shows how to build SGX software from sources, offering additional security guarantees for SGX’s authenticity.

Previous chapter looked at SGX’s hypothetical run-time system, this chapter focuses on SGX from hardware level to system level (figure 4). This is uncovered by practically setting up SGX environments that will form the basis for performance evaluation in the next chapter.

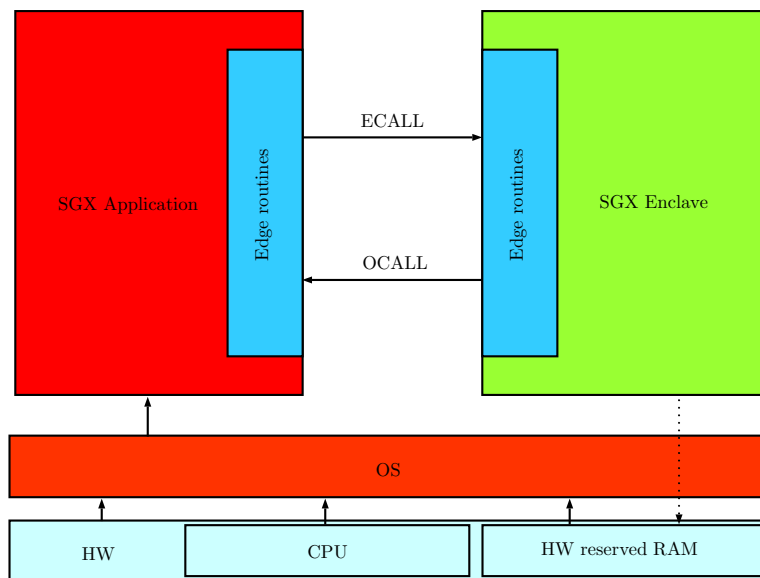


Figure 4. SGX from hardware level to system level.

3.1 SGX support

Determining SGX-capable hardware might not be the easiest thing. It is true that SGX lies within the Intel CPU, but the CPU alone doesn’t guarantee the system can run SGX.

SGX is supported on most modern Intel processors starting from 6th generation Core series. While the Core series is widely popular, SGX is also supported on Xeon, Pentium and Celeron series processors [45], [46]. SGX runs in conjunction with other Intel subsystems such as Intel Management Engine (ME), and in some cases, Server Platform Services (SPS) [45], [46].

To expose SGX to the OS, a capable system firmware (Basic Input Output System (BIOS)/Unified Extensible Firmware Interface (UEFI)) is needed [47]. Some system vendors opt not to enable this feature in the firmware, rendering the device unable to harness SGX [47].

SGX is supported on modern Windows and Linux distributions [48], [49]:

- Ubuntu 16.04 LTS 64-bit Desktop version
- Ubuntu 16.04 LTS 64-bit Server version
- Ubuntu 18.04 LTS 64-bit Desktop version
- Ubuntu 18.04 LTS 64-bit Server version
- Red Hat Enterprise Linux Server release 7.4 64bits
- CentOS 7.5 64bits
- Fedora 27 Server 64bits
- SUSE Linux Enterprise Server 12 64bits
- Microsoft Windows 7 64-bit version
- Microsoft Windows 10 November Update (version 1511) or later.

To harness SGX on the system-level, Intel provides a software stack consisting of SGX driver, SGX SDK and SGX PSW [50], [51].

Best method to check the hardware SGX support, is running the `cpuid` program, that checks for SGX from the CPU [52]. There is another, simplified version of this program, named `SGX-hardware` [47]. In a useful manner, `SGX-hardware`'s *readme* file maintains an up-to-date list of known SGX hardware [47].

3.2 Test hardware

Dell XPS 9560 laptop has SGX support from factory and is the testbed for two SGX environments. The first environment is host Linux, the second, guest Linux in VMM.

System Specifications:

- Intel Core i7-7700HQ,
- 16 GB DDR4 RAM @ 2400MHz,
- 512GB NVMe SSD,
- Arch Linux, Ubuntu 18.04.02,
- Intel ME disabled from factory,
- 128MB EPC,
- BIOS/UEFI version 1.14.2.

3.3 Linux system

Arch Linux 5.0.0-mainline was chosen for the native SGX environment, while it is not officially supported, it is demonstrated that it can efficiently run SGX software. This part serves as a practical guide on how to install SGX software.

The outline of work done is: installation of SGX driver, SGX PSW, SGX SDK and SGX Eclipse plugin. The order of installation is exact, and everything, except the Eclipse plugin, will be built from source. In this guide, SGX software version 2.5 is installed.

3.3.1 SGX driver

SGX driver sources are readily available on Intel's GitHub [50]. Driver build and installation is fairly standard procedure. The first thing is to acquire the sources (figure 5). Intel's SGX driver repository holds all version of the driver since v1.6 to current version.

```
$ git clone https://github.com/intel/linux-sgx-driver.git
$ cd linux-sgx-driver
$ git tag -l
$ git checkout sgx_driver_2.5
```

Figure 5. Fetch SGX driver.

Driver installation is a bit more complex procedure, where the ISGX module (driver) is installed into the current kernel (figure 6). It should be kept in mind that ISGX module is only tied with the current kernel and needs reinstalling if the kernel is changed.

```
$ make
$ sudo mkdir -p "/lib/modules/"$(uname -r)"/kernel/drivers/intel/sgx"
$ sudo cp isgx.ko "/lib/modules/"$(uname -r)"/kernel/drivers/intel/sgx"
$ sudo sh -c "cat /etc/modules | grep -Fxq isgx || echo isgx >> /etc/modules"
$ sudo /sbin/depmod
$ sudo /sbin/modprobe isgx
```

Figure 6. Install the ISGX driver module.

3.3.2 SGX SDK & PSW

Intel provides the full Linux SGX software stack on GitHub [51]. This repository is used to build and install the SGX SDK and SGX PSW. First, correct version of sources are acquired via git (figure 7).

```
$ git clone https://github.com/intel/linux-sgx.git
$ cd linux-sgx
$ git tag -l
$ git checkout sgx_2.5
```

Figure 7. Fetch Linux-SGX.

SGX SDK needs additional libraries which have to be downloaded using the provided script [51]. Install binaries can be built for different Linux distributions. In this case, standard binary packages are built both for the SGX SDK and PSW [51]. Built binaries can then be found in the bin directory (figure 8).

```
$ ./download_brebuilt.sh
$ make
$ make sdk_install_pkg
$ make psw_install_pkg
$ cd linux/installer/bin
```

Figure 8. Build Linux-SGX.

The installation procedure is fairly standard, but installation order is important. SGX PSW needs to be installed first, as it sets up the aesmd service, which starts system services linked with the SGX driver [51]. SGX SDK can then be installed. The installation path can be chosen by the user. In this case, the default path (/opt/intel) was used, which is also used by the SGX PSW and driver. After installation, the installer prints the environment variable for SGX SDK, which needs to be set by the user (figure 9).

```
$ sudo ./sgx_linux_x64_psw_2.5.100.49891.bin
$ sudo ./sgx_linux_x64_sdk_2.5.100.49891.bin
$ source /opt/intel/sgxsdk/environment
```

Figure 9. Install SGX PSW and SGX SDK.

The Eclipse plugin is available for download on SGX’s website [53], or it can be built from Linux-SGX sources [51]. The installation follows a standard Eclipse plugin installation procedure, detailed in SGX’s installation guide [48].

3.4 Virtualized Linux system

Intel provides preliminary patches to support SGX virtualization for Kernel Virtual Machine (KVM) and Xenial (XEN) [4]. These patches are for experimental purposes, and under active development [4]. SGX virtualization is also used in Graphene, Scone and Haven [4], [54].

This paper focuses on KVM-SGX and QEMU-SGX, which form the SGX environments used for performance evaluation. KVM-SGX [55] is a special Linux kernel that exposes SGX functions to an equally special VMM - QEMU-SGX [56] (figure 10).

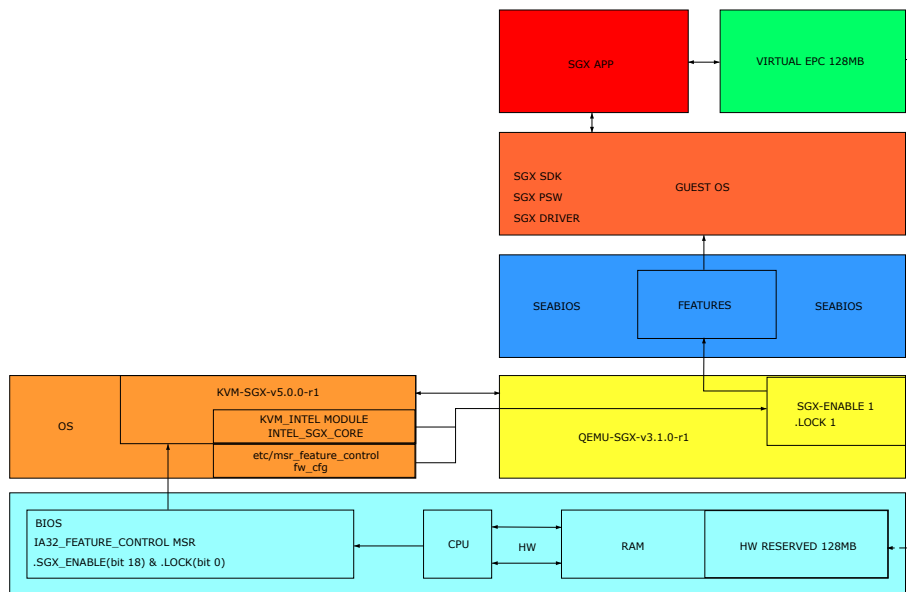


Figure 10. Working principles of KVM-SGX and QEMU-SGX.

3.4.1 KVM-SGX

KVM-SGX is a custom Linux kernel [56]. Depending on the Linux distribution, kernel building process might vary. In this paper, KVM-SGX v5.0.0-r1 is compiled for Arch Linux (figure 11) [57].

```

$ git clone https://github.com/intel/kvm-sgx.git
$ cd kvm-sgx
$ git tag -l
$ git checkout sgx-v5.0.0-r1

```

Figure 11. Fetch KVM-SGX.

KVM-SGX needs to be correctly configured. To build a reliable configuration, currently running Linux 5.0.0-mainline configuration is taken as the basis for the new kernel.

Configuration file needs some manual intervention, the name of the new kernel needs to be set, and two SGX modules should be enabled to expose SGX functions to the VMM [56]. When built, accompanying kernel modules must be installed, too, and the new kernel image must be copied to the boot directory (figure 12) [56].

```

$ zcat /proc/config.gz > .config
$ vi .config
-----
CONFIG_LOCALVERSION="-kvm-sgx"
CONFIG_INTEL_SGX_CORE=y
CONFIG_INTEL_SGX_DRIVER=y
-----
$ make LOCALVERSION=
$ sudo make modules_install
$ sudo cp -v arch/x86_64/boot/bzImage /boot/vmlinuz-linux-kvm-sgx

```

Figure 12. Configure and install KVM-SGX.

Finally, KVM-SGX ramdisk can be built that is used to initialize the new kernel. It is a fairly standard procedure, where a preset file is created containing needed boot configurations. This preset file can be based on the previous kernel's preset, but should match with the new kernel's file name in the boot directory (figure 13).

```

$ vi /etc/mkinitcpio.d/linux-kvm-sgx.preset
-----
# mkinitcpio preset file for the 'linux-kvm-sgx' package
ALL_config="/etc/mkinitcpio.conf"
ALL_kver="/boot/vmlinuz-linux-kvm-sgx"

PRESETS=('default' 'fallback')

#default_config="/etc/mkinitcpio.conf"
default_image="/boot/initramfs-linux-kvm-sgx.img"
#default_options=""

#fallback_config="/etc/mkinitcpio.conf"
fallback_image="/boot/initramfs-linux-kvm-sgx-fallback.img"
fallback_options="-S autodetect"
-----
$ sudo mkinitcpio -p linux-kvm-sgx

```

Figure 13. Make KVM-SGX ramdisk.

The new KVM-SGX kernel is ready for use.

3.4.2 QEMU-SGX

To take advantage of KVM-SGX features, an equally capable VMM is needed. QEMU-SGX is a VMM specifically designed to work with KVM-SGX [56]. QEMU-SGX contains patches, enabling options such as EPC management and allocation to the guest OS, VM migration, nested SGX virtualization, launch control, and explicit SGX feature and sub-feature controls [56].

QEMU-SGX's documentation states that KVM-SGX and QEMU-SGX only work together with specific software versions [56]. QEMU-SGX v3.1.0-r1 is compatible with previously installed KVM-SGX v5.0.0-r1 (figure 14) [56].

```
$ git clone https://github.com/intel/qemu-sgx.git
$ cd qemu-sgx
$ git tag -l
$ git checkout sgx-v3.1.0-r1
```

Figure 14. Fetch QEMU-SGX.

Regular Quick Emulator (QEMU) installation is quite complex by itself. To make the procedure of building and installing QEMU-SGX easier, a known good package of QEMU was taken as the basis, and QEMU-SGX was build and installed on top of it [58]. This method ensures that dependencies are satisfied, guaranteeing a successful default build. QEMU-SGX is a single executable binary that can be copied in place of the regular QEMU executable (figure 15).

```
$ ./configure --target-list=x86_64-softmmu
$ make
$ sudo pacman -S qemu
$ sudo cp -v x86_64-softmmu/qemu-system-x86_64 /usr/bin/qemu-system-x86_64
```

Figure 15. Build and install QEMU-SGX.

On the kernel level, QEMU-SGX additionally depends on KVM and Virtio modules, and Libvirt virtualization tools, API's and services (software) to launch QEMU-SGX.

KVM is a hypervisor built into the Linux kernel [59]. KVM is also a special operating mode of QEMU, capable of hardware-assisted virtualization (Hardware Virtual Machine (HVM)) [59]. KVM modules are available in Linux kernel since Linux 2.6.20 [60]. Two KVM modules need to be enabled in the host OS - *kvm* and *kvm_intel*, the second, specific to Intel CPUs [59].

Figure 16 demonstrates the procedure to check the CPUs VT-x (Virtualization Technology) support, and if KVM modules are present in the kernel (they can be disabled during compilation). Finally, modules are loaded and their state verified.

```
$ LC_ALL=C lscpu | grep Virtualization
$ zgrep CONFIG_KVM /proc/config.gz
$ sudo modprobe kvm
$ sudo modprobe kvm_intel
$ lsmod | grep kvm
```

Figure 16. Check for, and load KVM modules.

Virtio is a virtualization standard for network and disk device drivers that reveal their virtualized nature to the hypervisor [61]. Virtio enables high performance network and disk operations on the guest OS [61]. Virtio modules are usually available in the kernel or easily added at build time.

Figure 17 demonstrates the procedure to check for Virtio modules in the kernel, load them, and verify their state.

```
$ zgrep VIRTIO /proc/config.gz
$ sudo modprobe virtio-net
$ sudo modprobe virtio-blk
$ sudo modprobe virtio-scsi
$ sudo modprobe virtio-balloon
$ lsmod | grep virtio
```

Figure 17. Check for, and load Virtio modules.

The final piece of the puzzle is Libvirt. Libvirt is a collection of software that provides tools for easy management of VMs, virtual storage, and network interfaces [62]. QEMU-SGX needs the Libvirt daemon (figure 18) [62].

```
$ sudo pacman -S libvirt
$ sudo systemctl start libvirtd
```

Figure 18. Install and start Libvirt daemon.

3.4.3 Guest system

Guest OS was specifically chosen from the list of officially supported SGX OSs. Ubuntu 18.04.02 satisfies all requirements and is usually ready out-of-the-box [63]. This part demonstrates the capabilities of QEMU-SGX, and final preparation of the virtualized environment.

First, a disk image is prepared for the virtualized environment. A fairly long QEMU-SGX

command is used to boot from a live image of Ubuntu with attached devices needed to install the OS (figure 19) [64].

```
$ qemu-img create -f qcow2 Ubuntu 16G
$ qemu-system-x86_64 -boot d -cdrom ubuntu-18.04.2-desktop-amd64.iso -drive
file=Ubuntu -m 8192 -enable-kvm -cpu host -smp $(nproc)
```

Figure 19. Create a virtual disk and boot a live image of Ubuntu.

After installation, guest OS can be booted with SGX features enabled [56]. QEMU-SGX GitHub repository provides a sufficient manual to get SGX features up and running [56]. Through experimentation, it was determined that a maximum of 92MB can be allocated to the EPC.

QEMU-SGX needs explicitly configuration, (1) CPU, SGX and cache pass-through, (2) assign all available CPU threads, (3) create a static EPC of 92MB, allocate to the VM [56], [64], [65]. QEMU-SGX allows to partition the EPC and statically allocated it to the VM, or just map it, and not allocate it (figure 20) [56].

```
$ sudo qemu-system-x86_64 Ubuntu -m 8192 -enable-kvm -cpu host, +sgx,
host-cache-info=on -smp $(nproc) -object memory-backend-epc,id=mem1,size=92M
-sgx-epc id=epc1,memdev=mem1
-----
(1) -cpu host, +sgx, host-cache-info=on
(2) -smp $(nproc)
(3) -object memory-backend-epc, id=mem1, size=92M -sgx-epc id=epc1, memdev=mem1
```

Figure 20. Launch QEMU-SGX.

Lastly, SGX software stack is set up. The process is similar to the native Linux system. The only difference is that SGX PSW driver is installed through a .deb package [51]. The virtualized SGX environment runs the same versions of SGX software as the native Linux environment (figure 21).

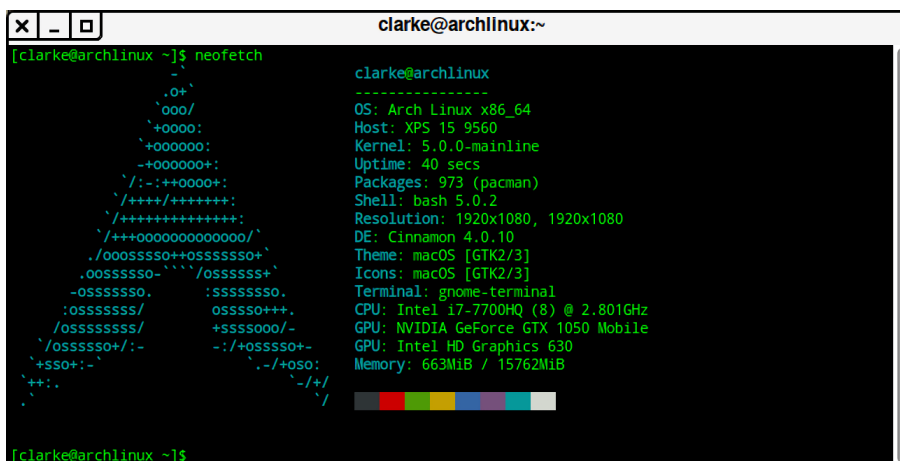
```
...
$ make deb_pkg
$ cd linux/installer/deb
$ sudo dpkg -i ./libsgx-urts_2.5.100.29891-bionicl_amd64.deb
./libsgx-enclave-common_2.5.100.49891-bionicl_amd64.deb
```

Figure 21. Install SGX PSW on Ubuntu.

3.5 Conclusion

This section covers SGX setup from hardware level to system level. It was found out that SGX support is not only determined by the CPU, but the system as a whole. This section introduced test hardware and planned environments. Setup procedure was described in detail. The result was three SGX systems, from which, the host and guest will be benchmarked in the next chapter.

System 0 is native Arch Linux 5.0.0-mainline with SGX support (figure 22). System 0 has mitigations for Spectre v1 and v2, Meltdown v3 and v4, Foreshadow-NG (OS) and Foreshadow-NG (VMM). Mitigations are not deployed for Meltdown v3a and Foreshadow (SGX).



```
clarke@archlinux ~]$ neofetch
      .o+
     'ooo/
    +oooo:
   +oooooo:
  -+oooooo+:
 /:-:++oooo+:
/++++/+++++++:
/+++++++:
/+++o000000000000/
./ooSSSSso++oSSSSso+
.oSSSSso-''''oSSSSs+
-oSSSSso.   ;SSSSSSo.
:osSSSSs/   osSSso+++
/osSSSSSS/  +SSSSooo/-
'/osSSSSo+!:-  -:/+osSSso+-
+SSo+:-      .-/+oso:
'.+::        .-/+:/
'.           .-/+:/
              .-/+:/

clarke@archlinux
-----
OS: Arch Linux x86_64
Host: XPS 15 9560
Kernel: 5.0.0-mainline
Uptime: 40 secs
Packages: 973 (pacman)
Shell: bash 5.0.2
Resolution: 1920x1080, 1920x1080
DE: Cinnamon 4.0.10
Theme: macOS [GTK2/3]
Icons: macOS [GTK2/3]
Terminal: gnome-terminal
CPU: Intel i7-7700HQ (8) @ 2.801GHz
GPU: NVIDIA GeForce GTX 1050 Mobile
GPU: Intel HD Graphics 630
Memory: 663MiB / 15762MiB
```

Figure 22. System 0: Arch Linux 5.0.0-mainline.

System 1 is a host Arch Linux 5.0.0-kvm-sgx, similar to system 0, but running a custom KVM-SGX kernel, which exposes SGX functions to the VMM (figure 23). System 1 has mitigations for Spectre v1, Meltdown v3 and v4, Foreshadow-NG (OS) and Foreshadow-NG (VMM). Mitigations are not deployed for Spectre v2, Meltdown v3a and Foreshadow (SGX).

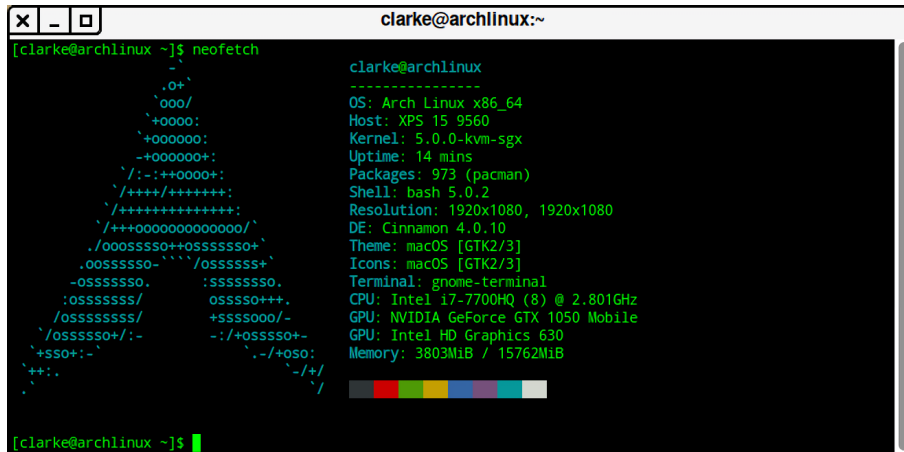


Figure 23. System 1, Host: Arch Linux 5.0.0-kvm-sgx.

System 2 is guest Ubuntu 18.04.02 Linux 4.18.0-17-generic running in VMM - QEMU-SGX, and supports all SGX features just like System 1 (figure 24). Guest has same speculative execution vulnerability mitigations as host.

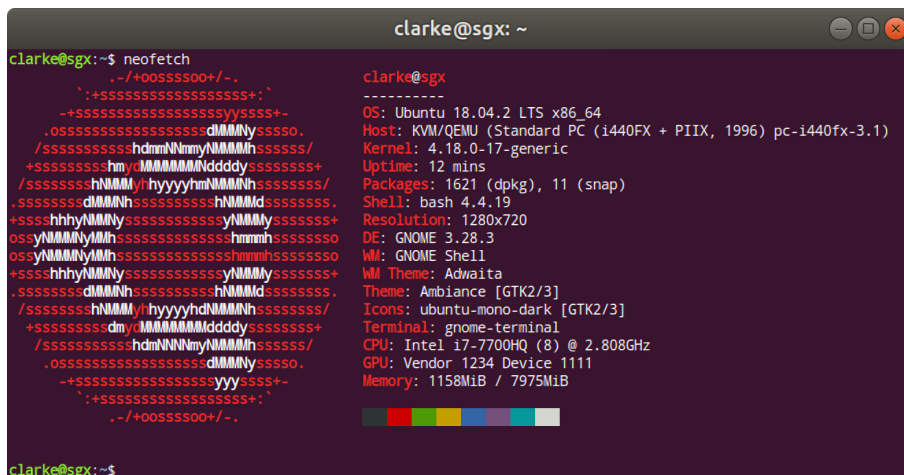


Figure 24. System 2, Guest: Ubuntu 18.04.02 Linux 4.18.0-17-generic.

4 Performance Evaluation

Intel SGX is a hardware based secure execution technology, which enables user-space (ring 3) code to create trusted memory regions named enclaves [66], [67]. The CPU defends the enclaves using a combination of memory access control and transparent memory encryption [67]. Enclaves are shielded from the OS, VMM, and SMM, offering a strong security guarantee, ensuring vulnerabilities and malicious code in any of these layers should not compromise the confidentiality and integrity of the secure-enclaves [66], [67].

The SGX hypothetical generic run-time system describes a typical SGX program. Although SGX programming paradigm is similar to conventional software, developers must follow the enclave programming model, which simply stated means partitioning the program into trusted and untrusted regions [1]. This raises the question on how to bring up rich applications on SGX, as some of them cannot be fully ported to run inside an enclave due to limited EPC size, and possibly exhibit higher system calls frequency [54], [66].

This brought on rapid development of library OSs and shims, such as Graphene-SGX, Scone, Haven and Panoply, which enable to quickly deploy applications in SGX, harnessing virtualization or API layers that wrap the application in an SGX-capable container [54]. Using a library OS or a shim is arguably impractical, both in performance overhead and TCB bloat, but research paper on Graphene-SGX demonstrates that these concerns are exaggerated, and that Graphene-SGX remains the best method to quickly bring up unmodified applications on SGX [54].

While research on porting applications to SGX has benefited the adoption of SGX technology, few have focused purely on the performance of edge routines - the primary interface between the trusted application and the untrusted enclave, remaining the primary cause of performance degradation [66].

To start evaluating SGX's performance, three edge routine methods were identified:

- Regular ECalls/OCalls [5];
- HotCalls [66];
- Switchless Calls [54], [5].

Regular ECall and OCall functions are the simplest of edge routine methods [5]. An enclave must expose an API for applications to call in (ECalls) and advertise what services provided by the untrusted domain are needed (OCalls) [1]. ECalls expose the interface that an untrusted application may use. By reducing the number of ECalls, enclave attack surface is reduced [1]. Enclaves have no control on which ECall is executed, or the order in which ECalls are invoked. Thus, an enclave cannot depend on ECalls occurring in certain order [1].

Enclaves cannot directly access OS-provided services. Instead, an enclave must make an OCall to an interface routine in the untrusted application. Calling outside adds a performance overhead without loss of confidentiality. Communicating with the OS requires release of data or the import of non-secret data, which needs to be handled properly [1].

Regular ECalls and OCalls, as specified by SGX developer reference manual, offer the full set of security guarantees of Intel SGX technology, although with performance implications [5], [1]. High performance overheads are associated with the number of assigned processing threads [5]. Regular ECalls and Ocalls are intrinsically single-threaded [5].

The first comprehensive quantitative study evaluating the performance of SGX, investigated the sources of performance degradation by leveraging a set of microbenchmarks for SGX-specific operations such as ECalls and OCalls, and encrypted memory I/O access. It was indeed discovered that the main loss in performance occurs due to expensive secure context switches and single threaded nature of regular SGX calls. The overhead of ECalls or OCalls is over 8000 CPU cycles, which is >50x more expensive than that of a system call (150 cycles). This degradation was remedied by a new SGX calls architecture and a synchronization spin-lock mechanism named HotCalls. They provide 13-27x speedup over the default interface [66].

SGX calls rely on expensive secure context switches. HotCalls proposed a requester/responder architecture that communicates via un-encrypted shared memory, employing a standby thread waiting for a call. It does so by polling a shared memory location. The shared memory is synchronized using a spin-lock. The requester - the enclave code makes a call and acquires the spin-lock and verifies that the responder - the untrusted code, is not busy. If the responder is available, the requester copies the relevant data to un-encrypted shared memory and points to that data via the **data* pointer. The code that encapsulates parameters within the *data* structure is the regular code automatically gener-

ated by the `edger8r` tool (regular ECall/OCall). Once the call ID and `*data` pointer are in place, the requester gives a go-signal to the responder by marking it as busy and releasing the spin-lock [66].

The proposed intricate HotCalls edge routines method indeed provided a boost in performance although with a trade-off between using a complicated programming model. HotCalls were proposed and implemented during the release of Intel SGX SDK v1.5 Beta, the first public release [66]. As of SGX SDK v2.2, this concept is improved upon, and a newer method named Switchless Calls is merged into the main SGX SDK branch [67], [68].

Switchless Calls use worker threads inside and outside of the enclave [5]. Switchless calls is a trade-off between assigning fixed (potentially wasted) resources to the application, and minimizing the performance penalty incurred by Regular ECalls/OCalls [67].

Research paper **Switchless Calls Made Practical in Intel SGX** set out to create a performance model that accounts for worker efficiency, static workloads and dynamic workloads [67]. Implementation of Switchless calls followed, with a focus on ease of use, robustness and customization of worker management [67]. It was shown that by using Switchless calls, latency compared to regular OCalls was reduced by 8.2x, and compared to ECalls, 5.9x [67]. Keeping in mind the trade-off, it was proven that Switchless calls are practical in short and frequent workloads, where they strike a balance between performance and energy conservation [67], [5].

This section evaluates the performance of three identified edge routine methods on host and guest environments. Different edge routine methods have practical and security implications, which will be evaluated along with their performance.

4.1 Testing methodology

Testing methodology is based on **Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves** research paper [66] and accompanying code [69].

Four code-bases are used for performance evaluation, from which, the first one is the official code from the authors of HotCalls [69]. The remaining code-bases are original work based on the HotCalls code and Intel's Switchless Calls SDK example.

Code-bases:

- 1. HotCalls, warm cache [69];
- 2. HotCalls, cold cache [70];
- 3. Switchless Calls, warm cache [71];
- 4. Switchless Calls, cold cache [72].

Each of the 4 code-bases consist of 4 microbenchmarks. Each microbenchmark executes 20 000 measurements via the RDTSCP function [3]. Measured round trip times of SGX call functions are expressed in clock cycles. Cold cache variants of code execute CLFLUSH function on data buffers before each measurement, evicting the data from every level of the cache hierarchy, ensuring that any modified data in the cache will be written to memory [3]. Data will be fetched from memory before each test execution [3].

Shell script is used to execute each microbenchmark 12 times in succession with 5 second pauses between them. First 2 sets of microbenchmark results will be discarded and remaining 10 sets will form the final dataset for each microbenchmark.

Microbenchmarks are targeting 2 previously set up systems running identical SGX software v2.5. System 1 is host Arch Linux 5.0.0-kvm-sgx, system 2 is guest Ubuntu 18.04.02 Linux 4.18.0-17-generic inside QEMU-SGX.

Correct benchmarking procedure was determined by experimenting with different launch configurations. Most consistent results were achieved by building each codebase in respective system and executing cold cache microbenchmark first, followed by the warm cache microbenchmark. To eliminate unwanted context switching, Intel Turbo Boost technology [73] was disabled by locking all cores to 2.8 GHz, networking also disabled.

Part of benchmarking was determining RDTSCP overheads for both systems. This was done by referring to Intel's code execution benchmark manual [74]. RDTSCP measurement tool was written in C++, which measures the average RDTSCP overhead of 200 000 function executions [75]. Measured average overheads were subtracted from respective systems' results. To manage processing the huge datasets, a collection of scripts and templates were written [76].

Final results consist of Cumulative Distribution Function (CDF) graphs, median cycles

and 99th percentile cycles. Final performance evaluation will be done using cold cache median results. 99th percentile results are used to further assess the tests. Due to redundant Regular ECall/OCall benchmarks, the worst performing median results will be used.

Hypothesis: System 1 outperforms System 2, and Switchless Calls outperform HotCalls.

4.2 Results

This section presents the full set of results for HotCalls and Switchless Calls. Results are used to compare System 1 vs. System 2, Regular Calls vs. HotCalls and Switchless Calls, HotCalls vs. Switchless Calls.

4.2.1 HotCalls

All conclusive measurement collected from warm cache and cold cache HotCalls code-bases are presented [69], [70]. Results are organized into a table for easy comparison of systems, cache types, and call types. Although warm cache results are not used for evaluation, they can be observed. 99th percentile results, marking the top 1% tests are also included (table 9). Measurements are plotted to graphs 25 and 26.

Table 9. Microbenchmark latencies in HotCalls.

#	Microbenchmark	System	Cache	Median latency (cycles)	99 th percentile (cycles)
1	Regular ECall	1	Warm	14 400	14 119
2	Regular ECall	2	Warm	15 379	15 159
3	Regular ECall	1	Cold	14 833	14 555
4	Regular ECall	2	Cold	15 768	15 544
5	Regular OCall	1	Warm	12 628	12 522
6	Regular OCall	2	Warm	13 480	13 293
7	Regular OCall	1	Cold	13 221	12 933
8	Regular OCall	2	Cold	14 110	13 787
9	Hot ECall	1	Warm	632	602
10	Hot ECall	2	Warm	3830	280
11	Hot ECall	1	Cold	1241	1064
12	Hot ECall	2	Cold	5335	766
13	Hot OCall	1	Warm	621	600
14	Hot OCall	2	Warm	3258	201
15	Hot OCall	1	Cold	1302	1029
16	Hot OCall	2	Cold	5212	733

Figure 25 demonstrates Regular Calls and HotCalls cycles on Systems 1. ECalls and OCalls (a) demonstrate consistent results. Same can be said about Hot ECalls and Hot OCalls (b), furthermore, the results are consistent with HotCalls research [66].

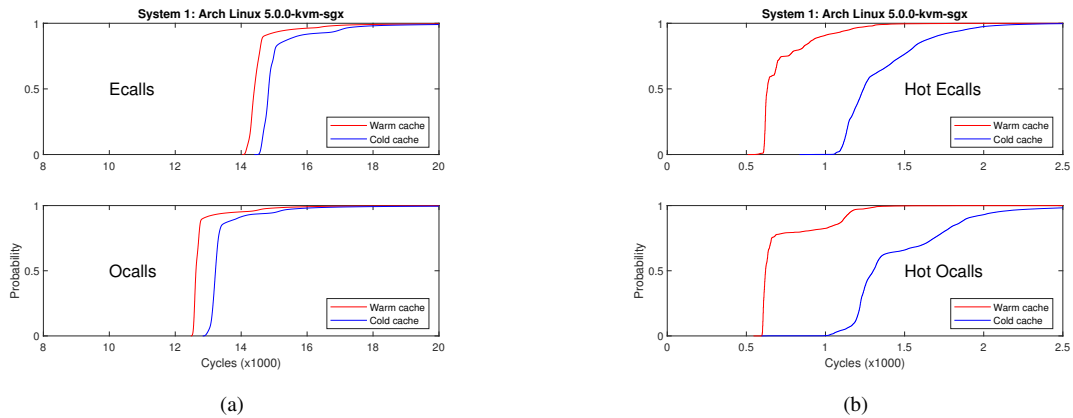


Figure 25. Test 1, HotCalls, System 1: (a) Regular Calls, (b) HotCalls.

Figure 26, System 2, demonstrates consistent results only in Regular ECalls and OCalls (a). HotCalls (b) exhibit undefined behavior.

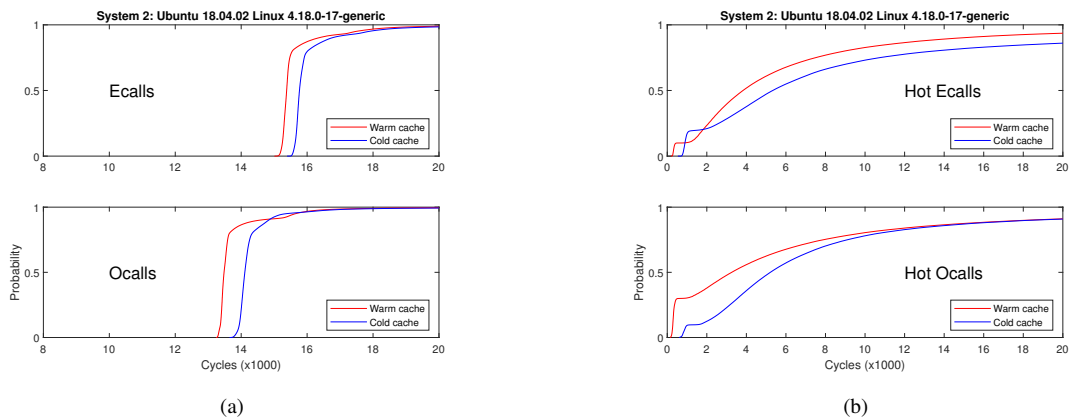


Figure 26. Test 2, HotCalls, System 2: (a) Regular Calls, (b) HotCalls.

Undefined behavior may be caused by experimental KVM-SGX or QEMU-SGX, but conclusive evidence is not found nor presented [56], [55]. Another explanation is that background context-switching (or something else) is forcing an AEX [66]. AEX was not monitored during testing. HotCalls research paper describes the same issue [66].

4.2.2 Switchless Calls

This section presents warm cache and cold cache measurement collected from Switchless Calls codebases [71], [72]. Results are presented in table 10 with plots on graphs 27 and 28.

Table 10. Microbenchmark latencies in Switchless Calls.

#	Microbenchmark	System	Cache	Median latency (cycles)	99 th percentile (cycles)
1	Regular ECall	1	Warm	14 224	13 976
2	Regular ECall	2	Warm	15 280	15 103
3	Regular ECall	1	Cold	14 411	14 172
4	Regular ECall	2	Cold	15 413	15 196
5	Regular OCall	1	Warm	12 557	12 433
6	Regular OCall	2	Warm	13 541	13 416
7	Regular OCall	1	Cold	13 107	12 834
8	Regular OCall	2	Cold	14 062	13 781
9	Switchless ECall	1	Warm	1426	1170
10	Switchless ECall	2	Warm	1421	1284
11	Switchless ECall	1	Cold	1582	1188
12	Switchless ECall	2	Cold	1484	1277
13	Switchless OCall	1	Warm	1172	670
14	Switchless OCall	2	Warm	1171	653
15	Switchless OCall	1	Cold	1757	1283
16	Switchless OCall	2	Cold	1732	1148

Figure 27 demonstrates Switchless Calls measurements on System 1. ECalls and OCalls (a) measurements are consistent with HotCalls codebase results. Switchless ECalls and OCalls (b) have a different characteristic than HotCalls, and are generally more consistent and stable. Results are consistent with Switchless Calls research [67].

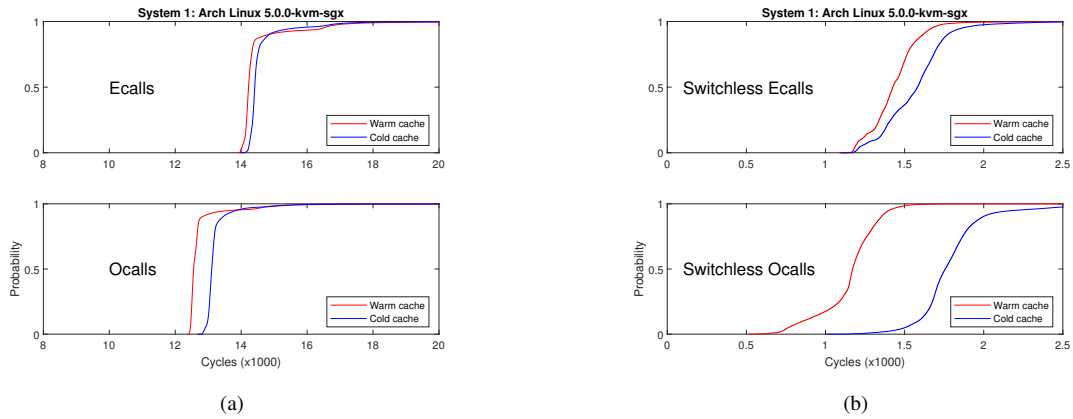


Figure 27. Test 3, Switchless Calls, System 1: (a) Regular Calls, (b) Switchless Calls.

Figure 28 demonstrates that Switchless Calls codebases on System 2 are also consistent and stable. ECalls and OCalls (a) results are consistent with HotCalls codebase measurements. Switchless ECalls and OCalls (b) perform equally fast to their System 1 counterparts.

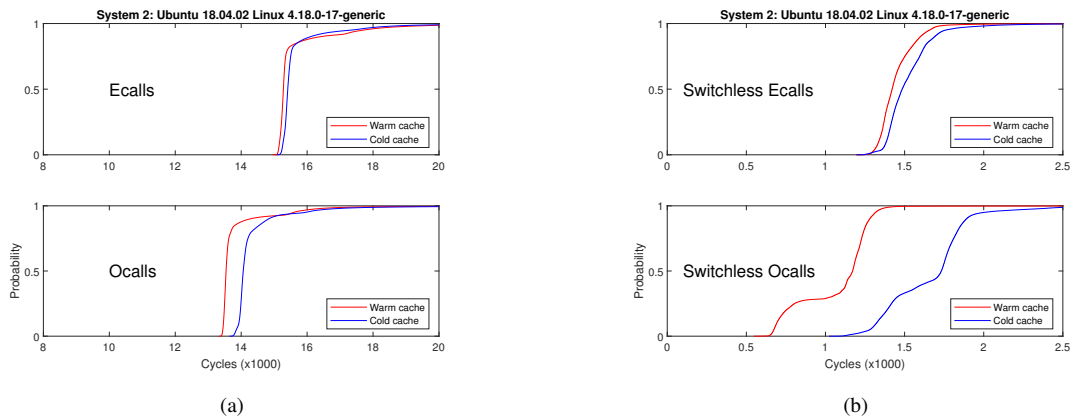


Figure 28. Test 4, Switchless Calls, System 2: (a) Regular Calls, (b) Switchless Calls.

Minor reduced latencies of Switchless Calls on System 2 should fall in the measurement error. The exact cause deserves further research.

4.2.3 Comparison

This section compares median results against each other to validate the Hypothesis. Comparisons will be made in three areas: System 1 vs. System 2, Regular Calls vs. HotCalls and Switchless Calls, HotCalls vs. Switchless Calls.

First area of interest is comparison of System 1 and System 2 overheads. System latencies are compared based on RDTSCP function (table 11), Regular Calls (table 12), HotCalls (table 13) and Switchless Calls (table 14).

Table 11 demonstrates the difference in RDTSCP overheads across two systems [75]. These results are subtracted from consecutive tests. Non-virtualized environment exhibits 1.2x speedup compared to the virtualized environment.

Table 11. System 1 vs. System 2 RDTSCP overheads.

Function	System 1	System 2	Latency Speedup
RDTSCP	32	37	1.2x

Table 12 demonstrates the difference in latencies of regular ECalls and OCalls across System 1 and 2. System 1 demonstrates 1.1x speedup in both cases.

Table 12. System 1 vs. System 2 Regular Calls overheads.

Function	System 1	System 2	Latency Speedup
ECall	14 833	15 768	1.1x
OCall	13 221	14 110	1.1x

Table 13 demonstrates the difference in latencies of HotCalls across System 1 and 2. System 1 shows 4.3x speedup in Hot ECalls and 4x speedup in Hot OCalls.

Table 13. System 1 vs. System 2 HotCalls overheads.

Function	System 1	System 2	Latency Speedup
Hot ECall	1241	5335	4.3x
Hot OCall	1302	5212	4.0x

Table 14 demonstrates the different in latencies of Switchless Calls across System 1 and 2. System 1 shows marginally higher latencies than System 2.

Table 14. System 1 vs. System 2 Switchless Calls overheads.

Function	System 1	System 2	Latency Speedup
Switchless ECall	1582	1484	1.1x
Switchless OCall	1757	1732	1.0x

Second area of interest is comparison of latencies of Regular calls to HotCalls (table 15, table 16) and Switchless Calls (table 17, table 18).

Table 15 shows 12x speedup in Hot ECalls compared to regular ECalls on System 1 and 3x speedup on System 2.

Table 15. Regular ECalls vs. Hot ECalls.

System	ECalls	Hot ECalls	Latency Speedup
System 1	14 833	1241	12.0x
System 2	15 768	5335	3.0x

Similar pattern continues in table 16, where Hot OCalls demonstrate a 10.2x speedup compared to regular OCalls on System 1, and 2.7x speedup on System 2.

Table 16. Regular OCalls vs. Hot OCalls.

System	OCalls	Hot OCalls	Latency Speedup
System 1	13 221	1302	10.2x
System 2	14 110	5212	2.7x

Very consistent results are achieved with Switchless ECalls on System 1, where 9.4x speedup is observed, and on System 2, where 10.6x speedup is observed, shown in table 17.

Table 17. Regular ECalls vs. Switchless ECalls.

System	ECalls	Switchless ECalls	Latency Speedup
System 1	14 833	1582	9.4x
System 2	15 768	1484	10.6x

Consistent results continue with Switchless OCalls, where 7.5x speedup is observed on System 1, and 8.1x on System 2, shown in table 18.

Table 18. Regular OCalls vs. Switchless OCalls.

System	OCalls	Switchless OCalls	Latency Speedup
System 1	13 221	1757	7.5x
System 2	14 110	1732	8.1x

Third area of interest is comparison of HotCalls to Switchless Calls (table 19, table 20).

Table 19 shows 1.3x speedup in Hot ECalls on System 1, and a 3.6x speedup in Switchless ECalls on System 2. 99th percentile latencies demonstrate that Hot ECalls perform better than Switchless ECalls, even on System 2.

Table 19. Hot ECalls vs. Switchless ECalls.

System	Hot ECalls	Switchless ECalls	Latency Speedup
System 1	1241	1582	1.3x
System 2	5335	1484	3.6x

The same pattern appears in table 20, where Hot OCalls demonstrate 1.3x speedup on System 1, and Switchless OCalls demonstrate 3x speedup on System 2. 99th percentile latencies once again place Hot OCalls first as the fastest performing of the two.

Table 20. Hot OCalls vs. Switchless OCalls.

System	Hot OCalls	Switchless OCalls	Latency Speedup
System 1	1302	1757	1.3x
System 2	5212	1732	3.0x

4.3 Conclusion

Three different edge routine methods - Regular Calls, HotCalls and Switchless Calls were benchmarked and compared in two SGX environments - host Arch Linux 5.0.0-kvm-sgx, and guest Ubuntu 18.04.02 Linux 4.18.0-17-generic running in VMM. An hypothesis was presented, which states that System 1 outperforms System 2, and Switchless Calls outperform HotCalls.

It was demonstrated that System 1 outperforms System 2 in every case, except with Switchless Calls, although with marginal difference. Using SGX in VMM adds a performance degradation up to 4.3x.

Comparison of Switchless Calls to HotCalls uncovered that HotCalls outperform Switchless Calls on System 1 with 1.3x speedup. System 2 demonstrated the opposite, where Switchless Calls outperform HotCalls with 3-3.6x speedup.

First part of the hypothesis can be considered confirmed, while the second part can not. Although...

The additional performance gains in HotCalls tests come at the expense of practicality and security. HotCalls are susceptible to AEXs, which leave them open to side channel attacks [36], [42]. AEX is the primary method used to force page-faults, completely breaking SGX's security guarantees [36], [42]. Another vulnerability is in threading, and can be exploited, e.g., with AsyncShock [27]. Additionally, HotCalls use un-encrypted shared memory to perform operations on secure data, completely defeating the purpose of SGX. Based on this, HotCalls implementation is not practical nor secure for SGX users.

HotCalls was the first fast calls method, and served as an important basis for Switchless Calls, which show negligible performance loss compared to HotCalls, but with added security. Switchless Calls conform to high security standards set for SGX.

Switchless Calls uses an innovative, efficiency based worker scheduling algorithm that determines the balance between allocating resources to the SGX application [67]. Switchless Calls are practical, stable, robust and highly customizable [67]. They are secured against known synchronization bugs [27]. They use secure multithreading by decoupling enclave and application threads [67]. Switchless Calls, along with Regular Calls, are part of the SGX SDK.

Based on measurements, and compared to a typical system call (150 cycles), Regular SGX Calls complete between 13 000 and 16 000 cycles, and cause a performance degradation of 88-105x in Intel SGX applications. This performance degradation can be reduced to 10x by using Switchless Calls, which complete between 1400 and 1800 cycles.

There are important findings in this section. KVM-SGX and QEMU-SGX show promising results in SGX virtualization, but their security and performance properties have not been thoroughly researched. This paper uncovered undefined behavior in edge routines running in VMM, and suggests additional research.

5 Summary

The goals of this thesis are to evaluate security and performance of SGX.

Security evaluation process encompassed extensive research on SGX from hardware level to software level with literature review of SGX vulnerabilities and exploits.

Performance evaluation encompassed research on SGX edge routine methods. Two SGX environments were set up, one of them virtualized, which served as benchmark environments. Microbenchmarking codes were based on HotCalls research and Intel's SGX SDK examples. Programming for microbenchmark codes included: writing a program to measure RDTSCP overheads and implementing RDTSCP-based measurement code into SGX applications. Variants of the codes were modified with CLFLUSH functions to take cold cache measurements. Extensive analysis and comparison of edge routine methods, and SGX environments followed.

This thesis presented four problems to solve: research on SGX architecture and security, SGX environments setup, programming microbenchmarks and performing benchmarks with conclusive results.

Research on SGX architecture and security covered the very basics of SGX up to complex real-world examples of known exploits. SGX uses a key derivation method stemming from CPUs hardware registers, transformed up to software level for subsequent derivations and sharing.

On the software level, SGX SDK uses IPP and SSL libraries equipped with modern encryption schemes to guarantee the confidentiality and integrity of data. Encryption library is accelerated by Intel AES-NI. There is an open debate about quantum computing breaking encryption schemes, most notably RSA and AES, also used in SGX. Researchers are calling out to create quantum-resistant algorithms today to replace the aging RSA and AES, which are nearing the end of their useful life-cycle.

Literature review presented case studies of successful attacks on SGX, where side-channel methods prevail. Intel publicly states that protection against all side channel vulnerabilities doesn't fall under the SGX security model. This is an open invitation to researchers and adversaries to completely break SGX in a wide variety of ways. Vulnerabilities that Intel mitigates are bugs in the software stack (including libraries) and speculative vul-

nerabilities like Foreshadow, widely associated with Spectre and Meltdown. The widest category of side-channel vulnerabilities are left to be mitigated by third parties.

Other vulnerabilities rise from intended mechanisms of SGX such as AEX, which is an enclave exit procedure performed by the CPU after an exception. Researchers discovered that AEX can be used to modify page tables and force a page-fault in the enclave which can be used to map its memory layout. It was later shown that an attack can be mounted without a page-fault, too.

The best mitigations implement Intel TSX to protect memory areas, successfully blocking side channels. ROP is the latest, arguably most dangerous attack method, which uses TSX to it's malicious intent to put undetectable malware into enclaves. Other ROP techniques use AEX to leak enclave secrets.

To close side channels on Intel CPUs, a new architecture is called for, which not only enhances the security of SGX, but of the whole platform.

SGX system setup covered the basis of finding SGX hardware and setting up environments. In this thesis, two environments were set up - host Arch Linux 5.0.0-kvm-sgx, and guest Ubuntu 18.04.02 Linux 4.18.0-17-generic running in VMM. Key components of SGX software stack were identified and successfully installed on Arch Linux. Virtualized system was built using KVM-SGX v5.0.0-r1 and QEMU-SGX v3.1.0-r1. The fairly complicated installation procedure was broken down.

Programming microbenchmarks and testing, covered extensive research into SGX edge routine methods. Three edge routine methods were identified: Regular Calls, Hot-Calls and Switchless Calls.

HotCalls, the first comprehensive quantitative study evaluating the performance of SGX, implemented a fast function calls method, providing significant speedup in SGX applications. This speedup came at the expense of security.

Switchless Calls, the official fast edge routine method, implemented into the SGX SDK v2.2, offered similar performance to HotCalls, but with added benefits of security.

Four code-bases are used for performance evaluation, from which, the first one is the official code from the authors of HotCalls. The remaining code-bases are original work based on the HotCalls code and Intel's Switchless Calls SDK example.

Programming is not extensively described in the thesis, but the codes are available on GitHub. The measurements were made with RDTSCP function, which measures CPU clock cycles. Cold cache code variants use CLFLUSH to flush data buffers before each test.

The comparison base is comprised of cold cache median results. Comparison was between two SGX environments, and between Regular Calls, HotCalls and Switchless Calls. The last comparison set comprised of HotCalls and Switchless Calls.

An hypothesis was presented, which states that System 1 outperforms System 2, and Switchless Calls outperform HotCalls. While this wasn't completely proven, few interesting outcomes were observed.

It was demonstrated that host SGX environment is 4.3x faster than the guest environment. When comparing Switchless Calls to HotCalls, it was demonstrated that HotCalls are 1.3x faster on System 1. The opposite was demonstrated on System 2, where Switchless Calls were 3-3.6x faster. When observing 99th percentile results, HotCalls outperformed Switchless Calls in every case.

The real conclusion in declaring the best fast calls method lies in security. HotCalls demonstrated vulnerabilities such as synchronization bugs, use of un-encrypted memory for secure data, defeating the purpose of SGX. Based on research and tests, Switchless Calls are declared the best method for fast SGX calls.

Based on measurements, and compared to a typical system call (150 cycles), Regular SGX Calls complete between 13 000 and 16 000 cycles, and cause a performance degradation of 88-105x in Intel SGX applications. This performance degradation can be reduced to 10x by using Switchless Calls, which complete between 1400 and 1800 cycles.

The important findings in this thesis are in KVM-SGX and QEMU-SGX. During testing, it was discovered that HotCalls and Switchless Calls perform differently from other tests in the guest environment. HotCalls research points out a similar issue, where background context-switching (or something else) is forcing an AEX.

Reduced Switchless Calls latencies in the guest environment may fall under measurement error, but equally surprising undefined behavior in HotCalls back up an hypothesis, that experimental KVM-SGX and QEMU-SGX hide undisclosed vulnerabilities.

References

- [1] Intel, *Intel® software guard extensions developer guide*. [Online]. Available: https://download.01.org/intel-sgx/linux-2.5/docs/Intel_SGX_Developer_Guide.pdf (visited on 04/20/2019).
- [2] V. Costan and S. Devadas, *Intel sgx explained*, Cryptology ePrint Archive, Report 2016/086, <https://eprint.iacr.org/2016/086>, 2016.
- [3] Intel, *Intel® 64 and ia-32 architectures software developer's manual - combined volumes: 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d and 4*. [Online]. Available: <https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf> (visited on 04/10/2019).
- [4] —, *Sgx virtualization | 01.org*. [Online]. Available: <https://01.org/intel-software-guard-extensions/sgx-virtualization> (visited on 04/10/2019).
- [5] —, *Intel® software guard extensions developer reference for linux* os*. [Online]. Available: https://download.01.org/intel-sgx/linux-2.5/docs/Intel_SGX_Developer_Reference_Linux_2.5_Open_Source.pdf (visited on 04/17/2019).
- [6] —, *Isca 2015 tutorial slides for intel® sgx*. [Online]. Available: <https://software.intel.com/sites/default/files/332680-002.pdf> (visited on 04/29/2019).
- [7] —, *Attestation service for intel® software guard extensions (intel® sgx): Api documentation*. [Online]. Available: <https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf> (visited on 04/28/2019).
- [8] —, *Introduction to intel® sgx sealing | intel® software*. [Online]. Available: <https://software.intel.com/en-us/blogs/2016/05/04/introduction-to-intel-sgx-sealing> (visited on 04/28/2019).
- [9] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, *Intel® software guard extensions: Epid provisioning and attestation services*. [Online]. Available: <https://software.intel.com/sites/default/files/managed/57/0e/ww10-2016-sgx-provisioning-and-attestation-final.pdf> (visited on 04/27/2019).
- [10] O. Wada and T. Namekawa, *Semiconductor electrically programmable fuse (efuse) having a polysilicon layer not doped with an impurity ion and a programming method thereof*. [Online]. Available: <https://patents.google.com/patent/US8279700B2/en> (visited on 04/27/2019).
- [11] Intel, *Intel(r) integrated performance primitives cryptography*. [Online]. Available: <https://github.com/intel/ipp-crypto> (visited on 04/28/2019).
- [12] —, *Intel® software guard extensions ssl*. [Online]. Available: <https://github.com/intel/intel-sgx-ssl> (visited on 04/28/2019).
- [13] NIST, *National vulnerability database - intel sgx*. [Online]. Available: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=Intel+SGX&search_type=all (visited on 04/23/2019).
- [14] —, *National vulnerability database - intel software guard extensions*. [Online]. Available: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=Intel+Software+Guard+Extensions&search_type=all (visited on 04/23/2019).

- [15] FIRST, *Common vulnerability scoring system v3.0: Specification document (v1.8)*. [Online]. Available: <https://www.first.org/cvss/cvss-v30-specification-v1.8.pdf> (visited on 04/24/2019).
- [16] A. D. Householder, G. Wassermann, A. Manion, and C. King, *The cert® guide to coordinated vulnerability disclosure*. [Online]. Available: https://resources.sei.cmu.edu/asset_files/SpecialReport/2017_003_001_503340.pdf (visited on 04/24/2019).
- [17] Intel, *Intel-sa-00117*. [Online]. Available: <https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00117.html> (visited on 04/23/2019).
- [18] —, *Intel-sa-00217*. [Online]. Available: <https://www.intel.com/content/www/us/en/security-center/advisory/INTEL-SA-00217.html> (visited on 04/23/2019).
- [19] —, *Intel-sa-00203*. [Online]. Available: <https://www.intel.com/content/www/us/en/security-center/advisory/INTEL-SA-00203.html> (visited on 04/23/2019).
- [20] —, *Intel-oss-10004*. [Online]. Available: <https://01.org/security/advisories/intel-oss-10004> (visited on 04/23/2019).
- [21] —, *Intel-sa-00161*. [Online]. Available: <https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html> (visited on 04/23/2019).
- [22] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, “Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution”, in *Proceedings of the 27th USENIX Security Symposium*, See also technical report Foreshadow-NG [23], USENIX Association, Aug. 2018.
- [23] O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom, “Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution”, *Technical report*, 2018, See also USENIX Security paper Foreshadow [22].
- [24] Intel, *Intel side channel vulnerability l1tf*. [Online]. Available: <https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html> (visited on 04/24/2019).
- [25] —, *Intel-sa-00076*. [Online]. Available: <https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00076.html> (visited on 04/23/2019).
- [26] —, *Academic research*. [Online]. Available: <https://software.intel.com/en-us/sgx/documentation/academic-research> (visited on 04/23/2019).
- [27] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, “Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves”, in *Computer Security – ESORICS 2016*, I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, Eds., Cham: Springer International Publishing, 2016, pp. 440–457, ISBN: 978-3-319-45744-4.
- [28] MITRE, *Common weakness enumeration - cwe-415: Double free (3.2)*. [Online]. Available: <https://cwe.mitre.org/data/definitions/415.html> (visited on 04/30/2019).

- [29] Y. Xu, W. Cui, and M. Peinado, “Controlled-channel attacks: Deterministic side channels for untrusted operating systems”, in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, ser. SP ’15, Washington, DC, USA: IEEE Computer Society, 2015, pp. 640–656, ISBN: 978-1-4673-6949-7. DOI: 10.1109/SP.2015.45. [Online]. Available: <https://doi.org/10.1109/SP.2015.45>.
- [30] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, “Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution”, in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, 2017, pp. 1041–1056, ISBN: 978-1-931971-40-9. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck>.
- [31] Y. Xiao, M. Li, S. Chen, and Y. Zhang, “Stacco: Differentially analyzing side-channel traces for detecting SSL/TLS vulnerabilities in secure enclaves”, *CoRR*, vol. abs/1707.03473, 2017. arXiv: 1707.03473. [Online]. Available: <http://arxiv.org/abs/1707.03473>.
- [32] A. Moghimi, G. Irazoqui, and T. Eisenbarth, *Cachezoom: How sgx amplifies the power of cache attacks*, Cryptology ePrint Archive, Report 2017/618, <https://eprint.iacr.org/2017/618>, 2017.
- [33] M. Hähnel, W. Cui, and M. Peinado, “High-resolution side channels for untrusted operating systems”, in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA: USENIX Association, 2017, pp. 299–312, ISBN: 978-1-931971-38-6. [Online]. Available: <https://www.usenix.org/conference/atc17/technical-sessions/presentation/hahnel>.
- [34] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, “Cache attacks on intel sgx”, in *Proceedings of the 10th European Workshop on Systems Security*, ser. EuroSec’17, Belgrade, Serbia: ACM, 2017, 2:1–2:6, ISBN: 978-1-4503-4935-2. DOI: 10.1145/3065913.3065915. [Online]. Available: <http://doi.acm.org/10.1145/3065913.3065915>.
- [35] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, “Malware guard extension: Using SGX to conceal cache attacks”, *CoRR*, vol. abs/1702.08719, 2017. arXiv: 1702.08719. [Online]. Available: <http://arxiv.org/abs/1702.08719>.
- [36] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiaainen, S. Capkun, and A. Sadeghi, “Software grand exposure: SGX cache attacks are practical”, *CoRR*, vol. abs/1702.07521, 2017. arXiv: 1702.07521. [Online]. Available: <http://arxiv.org/abs/1702.07521>.
- [37] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, “Inferring fine-grained control flow inside SGX enclaves with branch shadowing”, *CoRR*, vol. abs/1611.06952, 2016. arXiv: 1611.06952. [Online]. Available: <http://arxiv.org/abs/1611.06952>.
- [38] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, “T-sgx: Eradicating controlled-channel attacks against enclave programs”, in *NDSS*, 2017.
- [39] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, “Strong and efficient cache side-channel protection using hardware transactional memory”, in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, 2017, pp. 217–233, ISBN: 978-1-931971-40-9. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/gruss>.

- [40] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, “Spectre attacks: Exploiting speculative execution”, in *40th IEEE Symposium on Security and Privacy (S&P’19)*, 2019.
- [41] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, “Meltdown: Reading kernel memory from user space”, in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- [42] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang, “Hacking in darkness: Return-oriented programming against secure enclaves”, in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, 2017, pp. 523–539, ISBN: 978-1-931971-40-9. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk>.
- [43] M. Schwarz, S. Weiser, and D. Gruss, “Practical enclave malware with intel SGX”, *CoRR*, vol. abs/1902.03256, 2019. arXiv: 1902.03256. [Online]. Available: <http://arxiv.org/abs/1902.03256>.
- [44] R. Evers and A. Sweeny, *Reducing the time to break symmetric keys*. [Online]. Available: <https://kryptera.ca/paper/2018-03/> (visited on 05/03/2019).
- [45] Intel, *Intel® product specification advanced search*. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/search/featurefilter.html?productType=873&2_SoftwareGuardExtensions=Yes%20with%20Intel%C2%AE%20ME (visited on 04/10/2019).
- [46] —, *Intel® product specification advanced search*. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/search/featurefilter.html?productType=873&2_SoftwareGuardExtensions=Yes%20with%20both%20Intel%C2%AE%20SPS%20and%20Intel%C2%AE%20ME (visited on 04/10/2019).
- [47] L. Lühr, *Ayeks/sgx-hardware: This is a list of hardware which is supports intel sgx - software guard extensions*. [Online]. Available: <https://github.com/ayeks/SGX-hardware> (visited on 04/10/2019).
- [48] Intel, *Intel® software guard extensions installation guide for linux* os*. [Online]. Available: https://download.01.org/intel-sgx/linux-2.5/docs/Intel_SGX_Installation_Guide_Linux_2.5_Open_Source.pdf (visited on 04/10/2019).
- [49] —, *Intel® software guard extensions (intel® sgx) sdk for windows* os revision: 2.3 release notes*. [Online]. Available: <https://software.intel.com/sites/default/files/managed/d1/0a/Intel-SGX-SDK-Release-Notes-for-Windows-OS.pdf> (visited on 04/30/2019).
- [50] —, *Intel/linux-sgx-driver: Intel sgx linux* driver*. [Online]. Available: <https://github.com/intel/linux-sgx-driver> (visited on 04/10/2019).
- [51] —, *Intel/linux-sgx: Intel sgx for linux**. [Online]. Available: <https://github.com/intel/linux-sgx> (visited on 04/10/2019).
- [52] T. Allen, *Todd allen’s tools: Cpuid*. [Online]. Available: <http://www.etalen.com/cpuid.html> (visited on 04/30/2019).
- [53] Intel, *Intel sgx linux 2.5 release | 01.org*. [Online]. Available: <https://01.org/intel-softwareguard-extensions/downloads/intel-sgx-linux-2.5-release> (visited on 04/10/2019).

- [54] C.-C. Tsai, D. E. Porter, and M. Vij, “Graphene-sgx: A practical library os for unmodified applications on sgx”, in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC ’17, Santa Clara, CA, USA: USENIX Association, 2017, pp. 645–658, ISBN: 978-1-931971-38-6. [Online]. Available: <http://dl.acm.org/citation.cfm?id=3154690.3154752>.
- [55] Intel, *Intel/kvm-sgx*. [Online]. Available: <https://github.com/intel/kvm-sgx> (visited on 04/10/2019).
- [56] —, *Intel/qemu-sgx*. [Online]. Available: <https://github.com/intel/qemu-sgx> (visited on 04/10/2019).
- [57] A. Linux, *Kernel/traditional compilation - archwiki*. [Online]. Available: https://wiki.archlinux.org/index.php/Kernel/Traditional_compilation (visited on 04/10/2019).
- [58] —, *Arch linux - qemu 3.1.0-2*. [Online]. Available: https://www.archlinux.org/packages/extra/x86_64/qemu/ (visited on 04/10/2019).
- [59] —, *Kvm - archwiki*. [Online]. Available: <https://wiki.archlinux.org/index.php/KVM> (visited on 04/10/2019).
- [60] D. Marshall, *Kvm developer community gathers in first cross-industry event : @vmblog*. [Online]. Available: <http://vmblog.com/archive/2007/09/07/kvm-developer-community-gathers-in-first-cross-industry-event.aspx> (visited on 04/10/2019).
- [61] Libvirt, *Virtio - libvirt wiki*. [Online]. Available: <https://wiki.libvirt.org/page/Virtio> (visited on 04/10/2019).
- [62] A. Linux, *Libvirt - archwiki*. [Online]. Available: <https://wiki.archlinux.org/index.php/Libvirt> (visited on 04/10/2019).
- [63] C. G. L. [GB], *The leading operating system for pcs, iot devices, servers and the cloud | ubuntu*. [Online]. Available: <https://www.ubuntu.com/> (visited on 04/10/2019).
- [64] QEMU, *Qemu version 3.1.50 user documentation*. [Online]. Available: <https://qemu.weilnetz.de/doc/qemu-doc.html> (visited on 05/01/2019).
- [65] R. H. Bugzilla, *1428952 - enhance libvirt to present virtual l3 cache info for vcpus*. [Online]. Available: https://bugzilla.redhat.com/show_bug.cgi?id=1428952 (visited on 05/01/2019).
- [66] O. Weisse, V. Bertacco, and T. Austin, “Regaining lost cycles with hotcalls: A fast interface for sgx secure enclaves”, *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 81–93, Jun. 2017, ISSN: 0163-5964. DOI: 10.1145/3140659.3080208. [Online]. Available: <http://doi.acm.org/10.1145/3140659.3080208>.
- [67] H. Tian, Q. Zhang, S. Yan, A. Rudnitsky, L. Shacham, R. Yariv, and N. Milshten, “Switchless calls made practical in intel sgx”, in *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, ser. SysTEX ’18, Toronto, Canada: ACM, 2018, pp. 22–27, ISBN: 978-1-4503-5998-6. DOI: 10.1145/3268935.3268942. [Online]. Available: <http://doi.acm.org/10.1145/3268935.3268942>.
- [68] Intel, *Intel sgx linux 2.2 release | 01.org*. [Online]. Available: <https://01.org/intel-softwareguard-extensions/downloads/intel-sgx-linux-2.2-release> (visited on 04/17/2019).
- [69] O. Weisse, V. Bertacco, and T. Austin, *Oweisse/hot-calls: Fast interface for sgx secure enclaves. based on isca 2017 hotcalls paper*. [Online]. Available: <https://github.com/oweisse/hot-calls> (visited on 04/16/2019).

- [70] T. Lusmägi, *Sgx-performance/enclave-hotcalls-clflush*. [Online]. Available: <https://github.com/SGX-performance/enclave-hotcalls-clflush> (visited on 05/01/2019).
- [71] —, *Sgx-performance/enclave-switchless*. [Online]. Available: <https://github.com/SGX-performance/enclave-switchless> (visited on 05/01/2019).
- [72] —, *Sgx-performance/enclave-switchless-clflush*. [Online]. Available: <https://github.com/SGX-performance/enclave-switchless-clflush> (visited on 05/01/2019).
- [73] Intel, *Intel® turbo boost technology 2.0*. [Online]. Available: <https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html> (visited on 04/22/2019).
- [74] I. Gabriele Paoloni, *How to benchmark code execution times on intel® ia-32 and ia-64 instruction set architectures*. [Online]. Available: <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf> (visited on 04/22/2019).
- [75] T. Lusmägi, *Sgx-performance/syscall-and-rdtscp-overhead*. [Online]. Available: <https://github.com/SGX-performance/syscall-and-rdtscp-overhead> (visited on 05/01/2019).
- [76] —, *Sgx-performance/benchmark-files*. [Online]. Available: <https://github.com/SGX-performance/benchmark-files> (visited on 05/01/2019).

Appendix 1 – SGX instructions

Table 1 lists SGX1 instructions and leaf functions available since 6th generation Intel Core processors [3].

Table 1. Supervisor (ring 0) and User Mode (ring 3) Enclave Instruction Leaf Functions in Long-Form of SGX1 [3].

Supervisor instruction	Description
ENCLS[EADD]	Add a page
ENCLS[EBLOCK]	Block an EPC page
ENCLS[ECREATE]	Create an enclave
ENCLS[EDBGDR]	Read data by debugger
ENCLS[EDBGWR]	Write data by debugger
ENCLS[EEXTEND]	Extend EPC page measurement
ENCLS[EINIT]	Initialize an enclave
ENCLS[ELDB]	Load an EPC page as blocked
ENCLS[ELDU]	Load an EPC page as unblocked
ENCLS[EPA]	Add version array
ENCLS[EREMOVE]	Remove a page from EPC
ENCLS[ETRACK]	Activate EBLOCK checks
ENCLS[EWB]	Write back/invalidate an EPC page
User instruction	
ENCLU[EENTER]	Enter an Enclave
ENCLU[EEXIT]	Exit an Enclave
ENCLU[EGETKEY]	Create a cryptographic key
ENCLU[EREPORT]	Create a cryptographic report
ENCLU[ERESUME]	Re-enter an Enclave

Table 2 lists SGX2 instructions and leaf functions reserved for future Intel Core processors [3], known to be available in Intel NUC7CJYH and NUC7PJYH [47].

Table 2. Supervisor (ring 0) and User Mode (ring 3) Enclave Instruction Leaf Functions in Long-Form of SGX2 [3].

Supervisor instruction	Description
ENCLS[EAUG]	Allocate EPC page to an existing enclave
ENCLS[EMODPR]	Restrict page permissions
ENCLS[EMODT]	Modify EPC page type
User instruction	
ENCLU[EACCEPT]	Accept EPC page into the enclave
ENCLU[EMODPE]	Enhance page permissions
ENCLU[EACCEPTCOPY]	Copy contents to an augmented EPC page and accept the EPC page into the enclave

Table 3 lists SGX OVERSUB instructions and leaf functions reserved for future Intel Core processors [3].

Table 3. VMX Operation (ring 0) and Supervisor Mode (ring 0) Enclave Instruction Leaf Functions in Long-Form of OVERSUB [3].

Supervisor instruction	Description
ENCLV[EDECVIRTCHILD]	Decrement the virtual child page count
ENCLV[EINCVIRTCHILD]	Increment the virtual child page count
ENCLV[ESETCONTEXT]	Set virtualization context
User instruction	
ENCLS[ERDINFO]	Read information about EPC page
ENCLS[TRACKC]	Activate EBLOCK checks with conflict reporting
ENCLS[ELDBC/UC]	Load an EPC page with conflict reporting