November 3: SGX

Scribe: Marten Lohstroh

1 Introduction

Intel SGX (Software Guard Extensions) are instruction set architecture (ISA) extensions that enable **isolated execution** of binary code using hardware support of the processor. Isolated execution takes place in an **enclave**. As long as an attacker cannot tamper with the hardware, there are some security guarantees with regard to code that executes inside of an enclave. Specifically, the memory claimed by the enclave is encrypted with a key inaccessible to the operating system (OS). Although the OS may be able to corrupt data that is used by the enclave, it is not able to manipulate it in any meaningful way; a corruption would be detected upon reading the data inside of the enclave where it is decrypted and checked for integrity.

Hardware-enabled security mechanisms or attempts to provide a root of trust on the hardware level are not new; so-called Trusted Platform Modules (TPMs) have been developed for over a decade and are commonly available on current hardware platforms. A TPM can provide facilities for the secure generation and storage of cryptographic keys, generation of random numbers, etc. SGX, however, *is* very new. The SDK required to compile source code that leverages SGX is not publicly available. Microsoft has had the privilege to use Intel's SDK for the past two years, which explains why all scientific publications regarding SGX all originate from Microsoft. Berkeley will gain access to the SDK very soon. To date, we have only been able to learn about SGX from papers, but have not been able to acquire first-hand experience with SGX. Today's discussion is based on [3].

The main thrust of the development of SGX is digital rights management (DRM). Industry is interested in using the extensions for the "secure" distribution of copy-righted material, i.e., delivering content to users without allowing them to copy and share it afterwards. However, there are many other application areas that are of interest to researchers—in this lecture we discuss some of them.

2 Threat Model

The threat model assumes there is an attacker that is able to tamper with the entire software stack that is running on the machine (including root access to the OS), essentially controlling the machine. However, the attacker cannot subvert the hardware. In particular, the attacker cannot change how the processor works or extract the secret keys that are printed in the die. The processor is always trusted. The assumption about the attacker being unable to mount a hardware attack is quite reasonable in many scenarios since hardware attacks are easier to detect, more expensive, and much harder to achieve than software attacks.

3 How it works

The enclave provides isolation. Code and data can be loaded into the enclave while any code running outside of the enclave is unable to taint it. It is possible to run multiple enclaves on the same machine, and by default, they cannot access each other's code or data. In principle, anything stored in main memory that is used by the enclave is encrypted. Exchange of data between enclaves or with other applications is still possible through shared memory, but then the data must either be stored unencrypted, or the enclaves must setup a shared key. SGX provides hardware instructions for operations needed to securely execute code in enclaves. In total, SGX comprises seventeen instructions, each of which falls into one of the following categories:

- Enclave build and tear-down
  - Allocate protected memory for the enclave, load values into the protected memory, measure the values loaded into the enclave's protected memory, tear down the enclave after the application has completed.
- Enclave entry
  - Switch to "enclave mode", perform integrity checks, decryption, and encryption.
- Enclave exit
  - Clear processor state (scrub registers, cache).
- Security operations
  - Allow an enclave to prove to an external party that the enclave was built on genuine Intel hardware.
- Page instructions
  - Allow system software to securely move enclave pages to and from unprotected memory.
- Debug instructions.

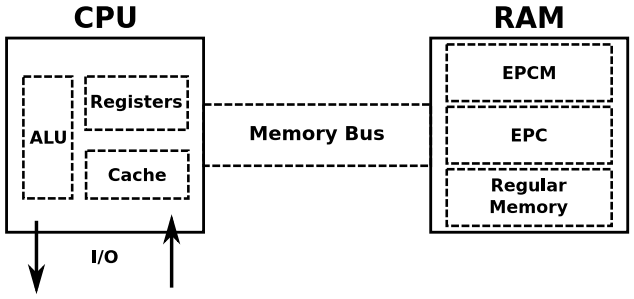

Figure 1: Schematic description of the CPU and RAM when operating in enclave mode.

SGX partitions the main memory in regions that are *protected* and memory that is generally accessible. A schematic of the SGX memory layout is depicted in Figure 3. It should be noted

that it is still the task of the operating system to swap pages in and out of regular memory. In addition, data used by enclaves needs to be swapped in an out of a protected memory region called the Enclave Page Cache (EPC), where enclave pages are stored. Enclave pages are 4KB long and encrypted using a special key that is kept inside the processor. Moreover, access to EPC is restricted on the hardware level. The Enclave Page Cache Map (EPCM), among other things, keeps track of which enclave is allowed to access which page(s) in the EPC; each page is mapped to a set of enclaves that can access it. The hardware checks whether a particular enclave is allowed to access a page based on the EPCM. Upon swapping out data, a keyed-hash message authentication code (HMAC) is generated to later verify both the data integrity and authenticity of the data when it is retrieved from unprotected main memory.

Access to the EPC is denied to unauthorized enclaves or processors that do not run in enclave mode. When the processor runs in "enclave mode" it appears to only use a single core, but details about the amount of concurrency that is possible in "enclave mode" are not discussed by Schuster et al. [3]. The reason for falling back to a single-threaded mode is that not only the registers, but also the cache lines are unencrypted, so a second process cannot be safely interleaved with the enclave code. Hence, exiting enclave mode also entails scrubbing the registers and cache to prevent leakage to other threads. However, based on [1] it appears that multi-threaded execution should be possible *within* an enclave.

3.1 A client-server application that leverages SGX

The client has code that it wants to run on the server on some private data, while keeping the code, the data, and the results confidential.

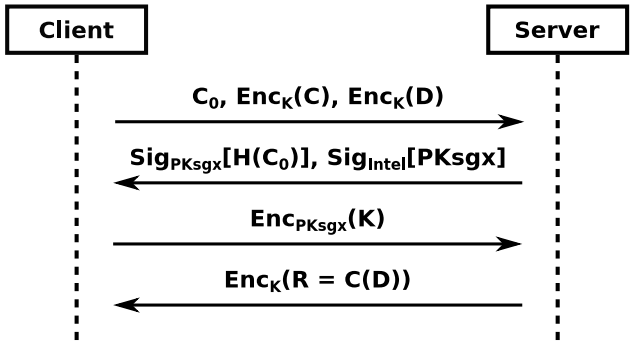

Figure 2: UML sequence diagram describing the exchange between client and server.

1. The client sends setup code (C_0)the server (not confidential), along with the code and the data, both encrypted with a secret key (K).

2. The server will setup an enclave and run $C0$ in it. A "measurement" is taken that captures the state of the machine (i.e., what software is currently running). As part of the attestation process, this measurement is encrypted by the SGX core and sent to the client, proving that C0 is currently running in isolation, inside an enclave.

3. After verifying the measurement (and the signature), the client can now trust the end-point

on the server as it runs in an enclave, hence it is safe to transfer the secret key to unlock the code and data. The client sends K, encrypted with the public key of the SGX core.

4. The setup code decrypts the code and data, runs C in a (separate) enclave and sends the result, encrypted with K, back to the client.

3.2 Remarks

A potential weakness of SGX is that an enclave could be occupied by malware, in which case it will be hard to detect.

4 Presentations

4.1 Verifiable Confidential Cloud Computing

Title: *VC3: Trustworthy Data Analytics in the Cloud Using SGX* [3]
Authors: Felix Schuster (Ruhr-Universitt Bochum), Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, Mark Russinovich (Microsoft Research)
Presenters: Jordan Kellerstrass, Jacob Lerner, Marten Lohstroh

4.1.1 Goals

The goal of VC3 is to provide shielded execution of distributed MapReduce computations in the Cloud in order to guarantee:

- Confidentiality of computation and results;
- Isolation of map and reduce computations;
- Integrity of data, computation, and code.

VC3 does not require the Cloud provider to be trusted. The technique, which relies on SGX, introduces only little overhead compared to Fully Homomorphic Encryption while achieving the same goal. Other tractable techniques may come at the cost of loss of generality (e.g., CryptDB only works on SQL databases), while VC3 has no bearing on the type of computation that it shields. VC3 aims to minimize the trusted computing base while safeguarding the confidentiality of computation, code, and data. It integrates easily with existing cloud software (most of it remains unchanged). A particular vulnerability remains despite the utilization of SGX enclaves; unsafe memory access while running in enclaved code. The authors address this issue by performing code analysis in their compiler toolchain to prevent unsafe memory access. The paper does not address DoS attacks, side-channels, or traffic sniffing.

4.1.2 Threat Model

The threat model assumes a malicious or compromised cloud provider, which may involve control over privileged software including hypervisors and firmware, a compromised management stack, etc. Data center employees or administrators or law enforcement is not trusted; they can be malicious or curious. The SGX processor is trusted. An attacker is unable to tamper with SGX-enabled processors in the data center. The user's code is benign, but it may not be perfect.

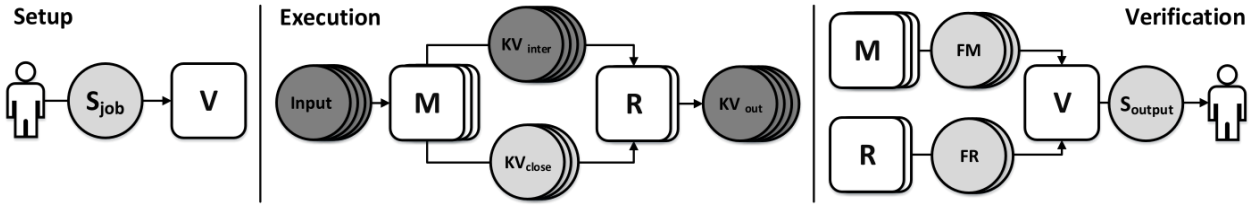

Figure 3: Schematic overview of our job execution protocol. The verifier (**V**), mappers (**M**), and reducers (**R**) are depicted as squares. A light-gray circle displays a message/key-value pair that is sent once by an entity; a dark-gray circle one that is sent multiple times. The user is depicted at both far ends. *Image and caption taken from Schuster et. al [3].*

4.1.3 Design Overview

Nodes that participate in a VC3 MapReduce computation perform a Map() or Reduce() function inside an SGX enclave. The code to be executed in the enclave is written in C++ and subjected to integrity checks in order to prevent unsafe memory access as it may result in information leakage. The code is compiled into a self-contained binary that must be able to run in the enclave. The code cannot rely on OS subroutines or depend on any libraries that do; it runs without any intervention from the OS and behaves much like a special purpose hypervisor.

The workflow for setting up a job is much like the sequence diagram in Figure **??**. Some public initialization code is used to establish a secure connection between the client and an SGX enclave that is used to receive the private code and data, and deploy the private code in its own enclave. The attestation procedure is slightly more elaborate as it additionally requires a signature from the Cloud provider to ensure that the machine that a job runs on is owned by a trusted organization. The idea behind this measure is that it would thwart complicated long-lived attacks that would involve extracting secrets from Intel hardware and inserting "compromised" SGX hardware into the pool of mappers and reducers.

Intermediate results that are sent as key-value pairs from mappers to reducers are encrypted; keys (bins) are obfuscated using a pseudorandom function (one-way hash) and values are encrypted using an intermediate key. Because the key-value pairs could be dropped or replayed, a *verifier* checks whether each pair is processed once and once only. Interestingly, the authors describe how this verification procedure can also be implemented as a MapReduce job as well.

4.2 Observing and Preventing Leakage in MapReduce

Title: *Observing and Preventing Leakage in MapReduce* [2]
Authors: Olga Ohrimenko, Manuel Costa , Cédric Fournet , Christos Gkantsidis, Markulf Kohlweiss (Microsoft Research), Divya Sharma (Carnegie Mellon University)
Presenter: Yi Wu

The paper addresses the issue of information leakage through traffic analysis. The authors point out that despite the use of encrption and secure hardware, computation such as MapReduce require shared resources (memory, I/O, network), thereby creating side channels. Under certain circumstances, the traffic between mappers and reducers can reveal sensitive information about

the data being processed. VC3 is targeted to stage a network traffic analysis attack against. It is assumed that an attacker can guess the particular job being executed (based on code size, data exchange profile, etc.) and may leverage statistical information about the input and output data of that job (e.g., distribution of marital status, residence state). For instance, the distribution of records across reducers may reveal that the data is sorted according to residence state; a reducer that works on records of people living in California will receive many more inputs than a reducer that is assigned records of people living in Virginia because California has considerably more inhabitants.

The solutions discussed in the paper involve padding and shuffeling of data. By shuffeling all the key-value pairs produced by the mappers before delivering them to the reducers, an adversery can still tell the volume of traffic for each reducer but cannot trace the traffic back to individual mappers. Additional dummy data can be sent to the less busy reducers to make all reducers appear to handle an equal workload, but this approach results in an unreasonable amount of overhead. The authors instead propose to first estimate the data distribution through random sampling, and then balance the workload by merging several keys into the same bin, letting less busy reducers take on the work for multiple keys.

References

[1] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 10:1–10:1, New York, NY, USA, 2013. ACM.

[2] O. Ohrimenko, M. Costa, C. Fournet, C. Gkantsidis, M. Kohlweiss, and D. Sharma. Observing and Preventing Leakage in MapReduce. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1570–1581, New York, NY, USA, 2015. ACM.

[3] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 38–54, 2015.