# USING INTEL® SGX TECHNOLOGIES TO SECURE LARGE SCALE SYSTEMS IN PUBLIC CLOUD ENVIRONMENTS

A Thesis
Presented to
The Academic Faculty

By

Jeffrey Edward Forster

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in
Computer Science

Georgia Institute of Technology

May 2018

Copyright © Jeffrey Edward Forster 2018

**USING INTEL® SGX TECHNOLOGIES TO SECURE LARGE SCALE SYSTEMS IN PUBLIC CLOUD ENVIRONMENTS**

Approved by:

Dr. Taesoo Kim
School of Computer Science
*Georgia Institute of Technology*

Dr. Wenke Lee
School of Computer Science
*Georgia Institute of Technology*

Dr. Ada Gavrilovska
School of Computer Science
*Georgia Institute of Technology*

Date Approved: December 21, 2017

# ACKNOWLEDGEMENTS

I would like to thank my advisor Taesoo Kim and the members of my committee for their time, patience and guidance. I would not have been able to do any of this without them. I would also like to thank Sandia National Laboratories for completely funding my pursuit of a master's degree. Finally, I would like to thank my parents Wendell and Lucille Forster for all the love and support that they have provided me throughout my life. I could not imagine life without them.

# TABLE OF CONTENTS

**Acknowledgments** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v

**List of Tables** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . x

**List of Figures** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xi

**Chapter 1: Introduction and Background** . . . . . . . . . . . . . . . . . . . . . 1

1.1 Problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.3 Trusted Execution Environment (TEE) . . . . . . . . . . . . . . . . . . . . 4

1.3.1 Rationale . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.3.2 Example Hardware and Uses . . . . . . . . . . . . . . . . . . . . . 5

1.3.3 Challenges . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**Chapter 2: Intel SGX** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2.1 Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2.1.1 Initialization . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2.1.2 Attestation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.1.3 Execution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.1.4 Ecalls and Ocalls . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.1.5 Overview . . . . . . . . . . 9
2.1.6 Setup . . . . . . . . . . 9
2.2 Benefits . . . . . . . . . . 11
2.3 Results . . . . . . . . . . 11
**Chapter 3: Graphene . . . . . . . . . . 13**
3.1 Description . . . . . . . . . . 13
3.1.1 Library Operating Systems . . . . . . . . . . 13
3.1.2 SGX Support . . . . . . . . . . 13
3.2 Advantages and Disadvantages . . . . . . . . . . 15
3.3 Results . . . . . . . . . . 16
**Chapter 4: Panoply . . . . . . . . . . 17**
4.1 Description . . . . . . . . . . 17
4.1.1 The Panoply Enclave Library . . . . . . . . . . 17
4.1.2 The Panoply Application Library . . . . . . . . . . 18
4.1.3 The Modified SGX Driver . . . . . . . . . . 18
4.1.4 Supporting System Calls . . . . . . . . . . 18
4.2 Usage . . . . . . . . . . 19
4.3 Benefits and Issues . . . . . . . . . . 20
4.4 Results . . . . . . . . . . 22
**Chapter 5: MySQL Security Overview . . . . . . . . . . 23**
5.1 Overview of Database Security Issues and Protections . . . . . . . . . . 23

5.1.1 Data Protections . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
5.1.2 Network Protections . . . . . . . . . . . . . . . . . . . . . . . . . 24
5.1.3 Operating System Protections . . . . . . . . . . . . . . . . . . . . 25
5.2 MySQL Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
5.2.1 Client . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
5.2.2 Network Component . . . . . . . . . . . . . . . . . . . . . . . . . 26
5.2.3 Query Processor . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
5.2.4 Storage Engine . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
5.2.5 System Library . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

**Chapter 6: Porting MySQL** . . . . . . . . . . . . . . . . . . . . . . . . . . 28
6.1 Threat Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
6.2 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
6.3 Porting MySQL with Graphene . . . . . . . . . . . . . . . . . . . . . . 29
6.4 Porting MySQL with Panoply . . . . . . . . . . . . . . . . . . . . . . . 29
6.4.1 Security Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
6.4.2 Configuring the Build . . . . . . . . . . . . . . . . . . . . . . . . . 30
6.4.3 Errors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

**Chapter 7: Related Work** . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
7.0.1 Secure Databases . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
7.0.2 SGX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

**Chapter 8: Conclusion** . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

8.1 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

**Appendix A: Setting Up System Calls as Ocalls** . . . . . . . . . . . . . . . . . . . 38

**Appendix B: MySQL Requirements not Supported By Panoply** . . . . . . . . . 40

**Appendix C: SGX CMake Scripts** . . . . . . . . . . . . . . . . . . . . . . . . 41

**References** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

## LIST OF TABLES

B.1 Functions and definitions used by MySQL not supported by Panoply . . . . 40

## LIST OF FIGURES


2.1 An overview of how an application is deployed and run using SGX. . . . . 9

3.1 An illustration of how Graphene supports system calls. . . . . . . . . . . . 14

4.1 The Panoply architecture and process. . . . . . . . . . . . . . . . . . . . . . . 19

# SUMMARY

Intel SGX enables securing applications at the hardware level, which makes it a very useful tool for running applications on untrusted hosts. Security based on hardware mechanisms is a rapidly evolving research area. SGX and similar tools have great potential to enhance security for many sensitive applications. Researchers are currently building tools that aid in porting of applications for use with the hardware based security. My work has demonstrated both benefits and limitations of the current SGX environment and associated tools. The current technology cannot support large applications such as MySQL server, and more work will need to be done to enable deployment of large-scale solutions.

# CHAPTER 1
# INTRODUCTION AND BACKGROUND

Moving computation to the cloud to virtualize hardware and centralize data is rapidly gaining in popularity [1]. Providers are able to get money for their unused resources, and clients are able to perform computations without needing the capital expenditure required for purchasing hardware and hiring maintenance staff. Using the cloud also allows for resource allocations to vary based on need, which makes it easier and cheaper to deal with traffic spikes based on events or the time of year [2]. This elasticity also allows for companies to grow rapidly without needing to worry about the power or space required for new servers.

While using cloud based services has many advantages, it does not come without risks. Virtual Machines (VMs) running on cloud networks have all the same risks that are associated with physical machines as well as some others because unlike a physical machine a VM is not isolated from other VMs. This means that virtual machines need to be protected from outside attacks as well as attacks from the same physical machine. Several issues have been raised about the safety of public clouds and virtualization in general [3].

## 1.1 Problem

Cloud based systems face a number of security threats from attackers, users, and even insiders. When the security of a cloud is compromised, data can can be lost, stolen or modified. Additionally, the service as a whole can become inaccessible or slow to the point of being unusable costing both the cloud clients and the cloud provider.

The Cloud Security Alliance (CSA) compiled a list of the top 9 most serious security threats that cloud systems need to protect themselves from. These threats are known as the ”Notorious Nine” [4]. The Notorious Nine are:

1. **Data breaches.** When storing data in the cloud, one of the most significant security flaws is the fact that, for the most part, all of the data for all of the customers is stored in the same location. Because of this grouping, it is possible for a secure system to be compromised if an attacker is able to successfully gain access to another less secure system. There is also threat that a malicious cloud provider employee wants to use his or her access to steal data from cloud clients [5].

2. **Data loss.** In some cases, even when data can't be accessed or stolen by an attacker, a vulnerability can lead to data being deleted or modified. Even a small loss of data can be catastrophic for companies that are storing their most valuable data in the cloud. Data loss can be caused by malicious users or by a simple operator error [6].

3. **Account hijacking.** Account hijacking is the act of exploiting vulnerabilities in an application to obtain a users private credentials. Account hijacking can be accomplished through cross-site request forgery vulnerabilities by tricking the user into opening an email or clicking a link which will run a set of malicious commands [7].

4. **Insecure interfaces.** In order to interact with, communicate with, and control the cloud, application programming interfaces (APIs) are often used as the intermediary interface for higher level systems. If these interfaces are weak or do not contain an adequate amount of security features in order to prevent exploits, the cloud as a whole may become susceptible to malicious activity.

5. **Denial of service attacks.** Denial of Services (DoS) attacks may be the simplest attacks to implement on a cloud computing device. A DoS attack consists of flooding an Internet connected device with requests. This overabundance of requests is harmful to the VM receiving them because the VM becomes overwhelmed with requests and therefore becomes unable to handle its standard request load. DoS attacks are very common and can bring even large systems down for extended periods. One

such DoS attack caused *BitBucket.org*, a popular code hosting site, to go down for 19 hours [8].

6. **Malicious insiders.** Any authorized individual who enters a cloud network with intentions of causing malicious actions is known as a malicious insider. Malicious insiders are able to apply a number of techniques to cause physical and financial damage to the businesses and institutions involved [5].

7. **Abuse of services.** One of the main advantages of cloud computing is that it is virtually computationally limitless. Cloud users have the illusion of having infinite computing resources available on demand [9]. Attackers can utilize these extensive computing resources to crack encryption keys, perform DoS attacks or perform other malicious attacks that would not be possible on limited hardware.

8. **Insufficient due diligence.** Many organization rush to move to the cloud for all the benefits without considering all the potential problems and threats. Even if a cloud provider has good protections from threats, there is a risk an organization's data could be compromised during the migration process [10]. Clients that rush to migrate to the cloud may also have misunderstanding about what the cloud provider will and will not do to protect them. These terms of service are specified in a provider's service level agreement (SLA) [11], and users often fail to understand the full scope of this contract.

9. **Shared vulnerabilities.** When handling, allocating, and managing a large set of virtual machines in the cloud, it is common to have multi-tenancy situations in which multiple machines share hardware resources. When VMs are co-located on the same system, a breach on one machine is capable of spreading to the other virtualized systems.

## 1.2 Motivation

There are many works that propose protections and solutions that can be applied to solve or alleviate the main threats related to cloud computing [12, 10, 13, 8, 14]. Cloud providers have many systems used to monitor and verify users access and identity. Access control and user verification systems can be used to reduce the threat posed by malicious users both internally and externally. Encryption and secure APIs can be used to help protect data during migration and while running in a virtualized environment.

While there are many different systems and solutions for the biggest cloud computing threats, the problem with many of them is that they require the cloud providers and customers to trust each other. The clients need to trust that the service provider

- is using secure technologies
- is not performing VM introspection to see or steal confidential data

The cloud providers, on the other hand, have to trust that the clients

- are not performing malicious or illegal activities using cloud resources
- are exercising due diligence in granting access to their systems
- implement effective security processes

The requirement for trust has made it impossible for large sensitive applications to be moved to public clouds.

## 1.3 Trusted Execution Environment (TEE)

One of the techniques that can be used to reduce the need for trust between a client and a cloud provider is the use of a secure area within a processor referred to as a trusted execution environment (TEE). TEEs allows for user-level processes to create and maintain a region of memory that is protected from breaches and modifications. These regions can

be referred to as enclaves, and they are inaccessible to any unauthorized process including higher level processes such as the operating system. This is accomplished through the use of software attestation where a process proves its identity using a secret cryptographic signature that is used in the creation of an enclave. If any unauthorized process tries to access a region of memory inside of a secured enclave, the processor will refuse to expose the data or allow for any changes to be made.

### 1.3.1 Rationale

Giving user level processes the ability to write and protect memory from any unauthorized access has many benefits for virtualized environments. Secure enclaves would allow for protections against three of the worst problems for untrusted public clouds: data breaches, malicious insiders and service provider side vulnerabilities. The extra security enclaves provide means that a client could run an application in a public cloud and know that the cloud provider would not be able to observe what the program was doing even with total control of the machine that the application was running on.

### 1.3.2 Example Hardware and Uses

Currently there are many processors that support protected trusted execution environments and also many different applications that use them. One example of a use of TEEs is the in the Apple iPhone's A7 chip. Apple's newer chips use secure enclaves for protecting fingerprints and mobile payments [15]. This gives the users an extra layer of security if their phone's OS is compromized. Another example of a hardware secured architecture is Intel's Software Guard Extensions (SGX). SGX is often used for small security tools such as key storage, password verification and digital rights management (DRM) [16].

### 1.3.3 Challenges

While using hardware secured enclaves can be a very useful layer of protection for many different tools and utilities, it cannot be easily applied to any application or problem. One of the main challenges for using enclaves is that they are limited to a very small portion of memory on the processor. This is not a problem when there is only a small amount of memory that needs to be protected such as a few cryptographic keys or a password, but when the memory that needs protected gets too large, other solutions have to be employed to ensure that the data stays confidential and unmodified.

Another challenge for using TEEs is that switching in and out of the enclave has a significant overhead because the processor has to verify that the process has permission to enter the enclave context. The verification overhead for an application that needs to enter and exit the enclave many times can greatly degrade performance.

Secure Architectures also have to limit the use of system calls because many system calls will expose the protected memory. For instance any unencrypted data written to a file by an enclave will be completely visible to other processes including the OS since enclaves protect memory but not disk space. All of these challenges make it difficult to create large scale systems that leverage hardware security methods.

# CHAPTER 2

# INTEL SGX

One of the main technologies that is currently being used for trusted execution environments is Intel Software Guard Extensions (SGX). SGX is a set of CPU instruction for the Intel architecture that allow for user processes to create enclaves that provide integrity and confidentiality guarantees on machines where all the privileged software is assumed to be malicious.

## 2.1 Architecture

SGX processors work by having a region of memory that is protected and referred to as the Processor Reserved Memory (PRM). This region is protected from any access attempts that are not made by an enclave process. Pages of the PRM region are managed by the CPU, which ensures that each page is assigned to no more than one enclave. Enclaves then have the ability to perform operations on their assigned PRM memory pages which are completely protected from any other process including the kernel, hypervisor and even other enclaves running on the same machine.

### 2.1.1 Initialization

To initialize an enclave, an unprotected system makes a request to the processor to allocate pages from the PRM. The system then makes a request to copy memory to the PRM. The requested PRM pages are then assigned to the new enclave and it is marked as initialized.

Because the enclave's state is initialized from unprotected memory, the initial enclave state will be known by the creating software. Once the enclave is initialized the CPU creates a cryptographic hash of all the contents inside of the enclave called a measurement hash. This measurement hash is used by the creating process to verify the identity and legitimacy

of the enclave.

### 2.1.2 Attestation

After an enclave is initialized a verifier can validate the identity and integrity of an enclave using a technique called software attestation. A verifier wanting to use software attestation asks the processor to produce an attestation signature. An attestation signature contains signed info about the data inside the enclave as well as a measurement hash of the software running inside the enclave.

The verifier can then use the attestation signature to check that the correct software created the signature and that the measurement matches. If the signature does not match, then the verifier knows that it is not communicating with the enclave. If the measurement does not match, the code inside of the enclave has been compromised.

### 2.1.3 Execution

In order for execution to enter the enclave context, special CPU instructions are set up. These special instructions allow for entering and exiting the enclave context as well as moving data in and out of the protected memory regions.

Once inside of an enclave there are limitations to the code that can be run. I/O functions are an example of operations that cannot be performed inside the enclave context. I/O functions are disabled because any data from the enclave that is written to the file system would be exposed to and compromise the confidentiality of the enclave.

### 2.1.4 Ecalls and Ocalls

Ecalls and Ocalls are the special CPU instructions that are used to move the execution of a program in and out of an enclave. Ecalls allow for user level code to authenticate itself and then access and execute the trusted code. Ecalls are considered trusted functions because after they are called all of the data and execution is protected. Ocalls are the opposite;

they are called from inside of the enclave to return to the unprotected area of code. Ocalls are considered untrusted because any data or code that is executed by an ocall could be inspected or modified by an adversary. These two types of functions allow for remote systems to access and control enclaves that could be located on untrusted machines.

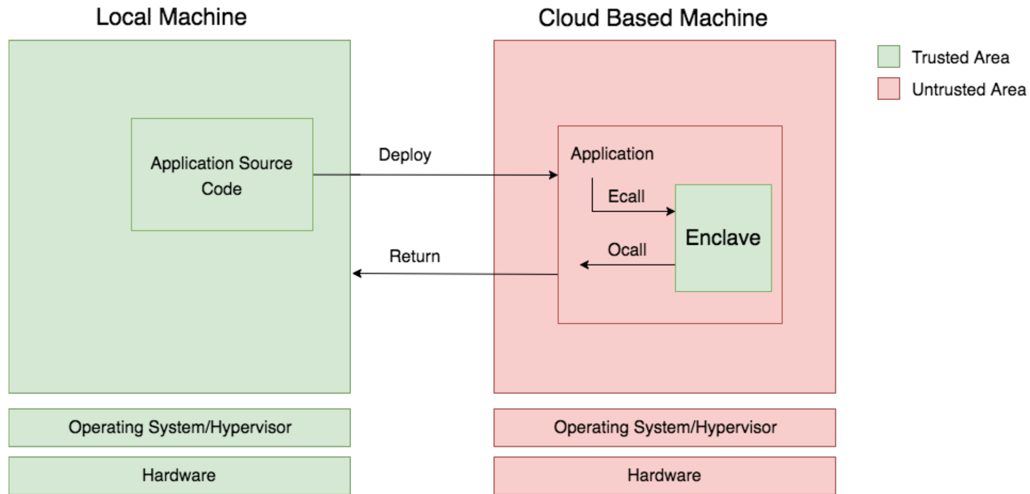


Figure 2.1: An overview of how an application is deployed and run using SGX.

### 2.1.5 Overview

Figure 2.1 shows a basic flow of how an application is deployed and run. It also illustrates which components are trusted and unsafe and how to switch between contexts with ecalls and ocalls.

### 2.1.6 Setup

Ecalls and Ocalls are defined using a special syntax called the Enclave Definition Language (EDL). EDL is used to specify the names and types used by all ecalls and ocalls. These functions can contain any basic C/C++ types such as int, char and short as well as pointers to more complex data types. EDL definitions are slightly different than standard C/C++ definitions in that they require additional information about the size of the parameters and whether parameters will be going into the enclave or coming out or both. The extra in-

formation is used by the processor to determine how much space to allocate and also to prevent unnecessary data leaks.

Code 2.1: Example EDL file

```
enclave {
    include "sys/types.h"

    from "extra_functions.edl" import *;

    /* Trusted ecall functions that will be executed inside enclave */
    trusted {
        int example_ecall1([in, string] char* str);
        int example_ecall2(size_t len, [out, size=len] void* buffer);
    };
    /* Untrusted ocall functions that will be executed outside enclave */
    untrusted {
        int example_ocall1([user_check] void* data);
        int example_ocall2(int number, [in, out, size=20] void* buffer);
    };
};
```

EDL files contain an enclave definition. The enclave definition begins with any includes and imports. Includes are normal C/C++ header files that can be used to define types used in function definitions. Imports are used to include other definitions from other EDL files. In Code 2.1 all trusted and untrusted functions are imported from a file called `extra_functions.edl`. After the imports and exports there are sections for trusted and untrusted functions. The trusted functions section is used to define protected ecalls, and the untrusted section is used to define ocalls.

Each pointer parameter in a function definition is preceded by options wrapped in square brackets. The in option is used to specify that a value is passed into the function, while the out option is used to state that the pointer is returned to the caller. If both of these options are used as in `example_ocall2()` from Code 2.1, the pointer value will be used as an input and return value. In addition to the in/out options a size can also be specified. The size can be a constant value, another integer parameter or if the object is null terminated, it can be specified as a string. If a user does not want the processor to provide any extra protections for a pointer, the user_check option can be used.

The implementations for all ecalls and ocalls will be done in normal C/C++ code. Ocalls are implemented in the unprotected code regions and will have access to all the features built into the processor including system calls and all the standard definitions and functions. Ecalls will be implemented inside of the protected enclave and will be limited in the types of functions that can be called.

## 2.2 Benefits

SGX provides all of the advantages of using a trusted execution environment for cloud based system. Organizations wishing to move their systems to the cloud using secure hardware may select SGX for a number of reasons. Many current systems use SGX capable hardware, which means clients have many choices when it comes to providers. The availability of multiple providers reduces a clients' dependence on a single provider to meet all of its needs as the system expands and changes over time.

Another advantage of Intel SGX is that development is relatively straightforward. The EDL language is very familiar to anyone with C/C++ experience and all of the protected code uses the standard C family of languages.

## 2.3 Results

SGX is becoming widely available and provides a level of protection that was not possible with older technologies. Many groups and organizations are developing new tools that take advantage of the hardware protections SGX can provide. Most of the SGX products that exist today are small tools that add an extra level of security for protecting cryptographic keys or other small bits of sensitive data.

There are two main reasons for the existing products being small utilities. The first reason is that the protected memory region on an SGX chip is very small. It works perfectly for a key or a fingerprint or other information that can be used for validation and verification, but there is no easy way to actually process large scale data such as a database. The

second primary reason is that there is no straightforward way for an existing system to be converted to use SGX capabilities. Existing systems do not by default separate the protected and unprotected regions of code and data. Furthermore, the protected code cannot take advantage of any system calls.

These two issues make it very difficult to utilize enclaves to add protection to large scale systems. For smaller applications, it is not too difficult to remove or replace system calls from protected regions and setup which functions need to be protected and which don't. However, for large applications with hundreds of thousands of lines of code, the process is non-trivial. Major applications can be cross platform and make use of hundreds of system calls across different platforms. Large systems may also have many different components and it can be difficult to assess which components need protections and how they should share data between enclaves or with shared enclaves.

Because of these issues, several tools have been developed to make migrating an existing project into SGX much less of an engineering task. These tools help to deal with system calls and try to make it easier to separate the code into protected and unprotected areas. Even with these partially automated tools, migrating an existing large scale system to SGX has still not been completely demonstrated.

# CHAPTER 3

# GRAPHENE

A tool that has been developed to help alleviate difficulties with SGX migration is called Graphene. Graphene is a Linux tool that is intended to allow for running unmodified Linux applications inside of SGX [17]. Graphene is able to support unmodified applications by emulating Linux system calls as an application library that can be run inside of an enclave.

## 3.1 Description

### 3.1.1 Library Operating Systems

Graphene is a Library OS project. Graphene allows for running single and multi-process applications in isolated environments. Library OSes work by refactoring a traditional kernel into an application library.

Library OSes provide a number of advantages over conventional operating system designs. They use a single address space, which can reduce the memory footprint by a significant amount. Library OSes do no need to move data between the user and kernel space, which removes the overhead of repeated privilege transitions.

Library OSes also have their disadvantages. Managing the shared and isolated areas can become complex when many applications are running side by side. In addition, Library OSes are closely tied to the hardware, which means they need to be regularly updated and rewritten because hardware changes rapidly.

### 3.1.2 SGX Support

In addition to all of the usual benefits of using a Library OS system, Graphene also supports multi-process applications and SGX. Graphene is able to run unmodified applications

inside of enclaves because the Graphene library and the application can both be loaded into an enclave. With the library inside of the enclave there is no need to utilize the base operating system to handle system calls.

Developers can run unmodified Linux applications using Graphene with minimal effort. The first step is to create a plain-text configuration file called a manifest file. The manifest file contains information about the environment and resources that the application will use. The manifest specifies what executable to use, what libraries need to be loaded and which environment variables will be needed. The manifest file is used by the loader. The manifest also includes system and network information such as stack size, mount points and network connections that will be made.

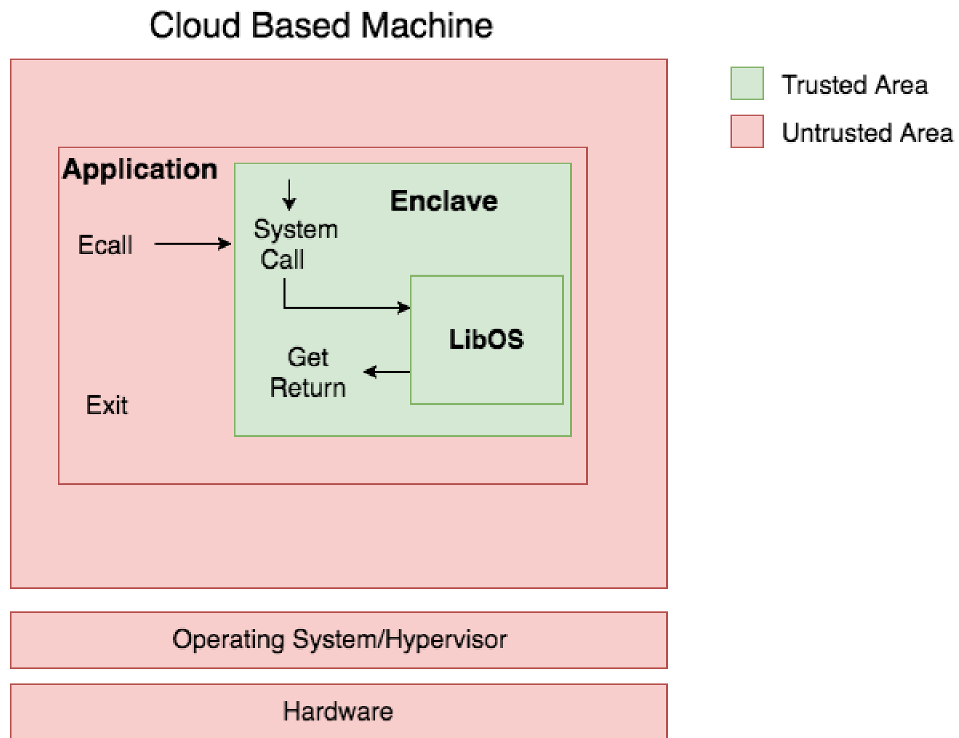


Figure 3.1: An illustration of how Graphene supports system calls.

After the manifest file is created a developer will need to build the Graphene libraries, and then sign the application using the signing tool that is built into Graphene. The signing tool will create an enclave signature and an SGX specific manifest file. After all of these

steps are complete, the developer can then ship there application files, signatures, libraries, and manifest files to the untrusted remote machine. Figure 3.1 illustrates how applications built using graphene run on a remote machine.

### 3.2 Advantages and Disadvantages

Graphene has multiple advantages over the standard methods of porting an application to SGX. Porting using Graphene requires much less engineering and does not require modifying systems at all. Because of this, Graphene applications can be moved between trusted machines and untrusted machines without needing to reconfigure anything or maintain different versions. Graphene removes the need for a developer to determine what components and data need to be protected because Graphene doesn't involve the operating system or expose any of the applications data without encryption.

Graphene and other Library OS solution have limited hardware support. Because Library OSes are so closely tied to hardware, it is not easy to support all the different systems that cloud providers may use. This could lead to problems with maintanence if the Library OS does not support upgrades to hardware.

Another potential issue is that Graphene is all or nothing with respect to protected application regions. If an application only has a small segment that is sensitive, there will be significant overhead used in protecting data and code that does not need it. To actually separate protected and unprotected regions of an application with Graphene, the developer would need to split the system into multiple applications with some method of communication between them.

A final disadvantage of using Graphene is that as of now, only about a third of Linux system calls are fully supported by the Graphene library. The rest are either partially supported or just return an error. If any of these unsupported system calls are used by an application, the developer wishing to port the application would need to either implement them by him self or modify the application to not use those calls. Neither of these solutions

are ideal because they both require extra testing and can become maintenance issues when either the application or Graphene is updated.

## 3.3 Results

The simplicity of running applications in SGX using Graphene has allowed for many applications to be ported. Graphene has made it possible for larger applications to utilize the hardware encryption protections that Intel has created into their processors. Some of the applications that have been implemented using Graphene are LMBench, Python, Apache and OpenJDK 1.7 [17].

# CHAPTER 4

# PANOPLY

Another tool that has been released to assist in the development and porting of SGX application is called Panoply. Panoply exposes the operating system functions to the enclave, which allows for applications with system calls to be executed. The SGX model assumes that the operating system is malicious, so no system calls are permitted. When implementing a panoply application, the developers will have to ensure that system calls do not expose any sensitive data. Because the system is untrusted, Panoply includes checks to ensure that the values returned by the operating system are consistent with what is expected. The Panoply features allow for applications to be run inside of SGX with limited modifications.

## 4.1 Description

Panoply allows for applications to be easily be compiled inside of an enclave with limited modification by allowing the untrusted operating system to perform actions that the secure hardware cannot. It is able to outsource the unsupported functions to the operating system through three separate components. Each of these components has its own role in the process and each can be modified or replaced independently of the others.

### 4.1.1 The Panoply Enclave Library

The first component of the Panoply system is a library that lives inside the enclave. This library defines all the Linux system calls as untrusted ocalls. The enclave library is also responsible for verifying the output from the operating system, since it is assumed to be malicious.

The enclave library performs basic checks that the output returned by the system is in an expected range and that nothing has changed that should not have. The calls in the

Panoply library will use the same definitions as the standard libraries system calls. This allows for Panoply calls to be easily incorporated into existing systems by removing links to the standard library and pointing to the Panoply enclave library.

### 4.1.2 The Panoply Application Library

The second component of the Panoply tool is the application library. This component lives in the untrusted region of a system. The application library's function is to provide a wrapper for all necessary system calls. It calls the system functions and returns the response back to the enclave code. Since all this code is unprotected from modification/deletion by privileged systems, no additional security checks are performed at this level.

### 4.1.3 The Modified SGX Driver

The final component of Panoply is a modification to the Linux SGX driver and SDK. Panoply requires this modification to be installed to the host system to support multi process applications, because the default SGX driver does not allow for multiple processes to share enclave memory which means that fork calls will result in a memory mismatch for parent and child processes. The developers of Panoply overcame this issue by ensuring that the virtual address space for all child processes begins at the same location as the parent [18].

The modified driver is an optional component that only needs to be installed if the application to be ported uses fork calls to run multiple processes that need to share memory. Using standard driver and SDK from Intel, single process applications will still function as expected if ported using Panoply.

### 4.1.4 Supporting System Calls

When an application makes a request for a system call to be executed from inside of an enclave, the enclave library receives the request and makes a call to the application library.

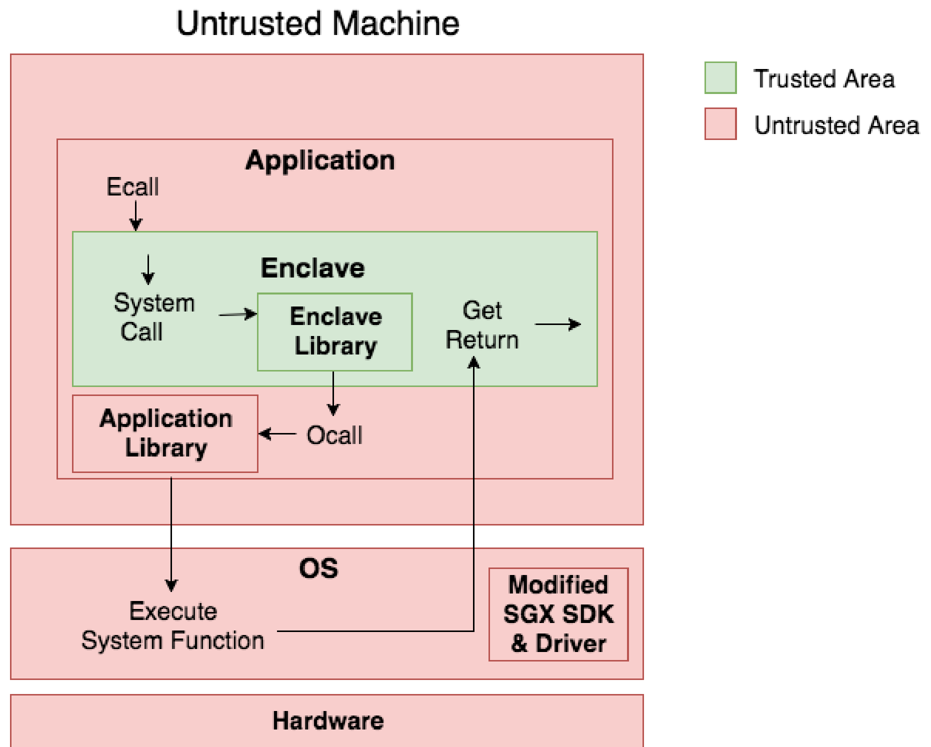


Figure 4.1: The Panoply architecture and process.

The application library will then send the request to the operating system and return the result. Figure 4.1 illustrates this process as well as showing where each of the three components is located on a machine.

## 4.2 Usage

There are a series of steps required, In order to port an application using Panoply, the first step is to determine which components of the application need to be protected and if they use fork calls. If fork calls are used, the host system needs to have the modified SDK and driver installed. The driver installation step will need to be reapplied each time the application is moved to a new machine or the program will not function as expected.

After the machine has been set up and verified, the application can then be modified to use the Panoply libraries. In order to make the application use the Panoply libraries instead of the standard OS libraries, all references to the standard libraries must be removed from

the source code and build files. Panoply ships with modified header files that include type and function definitions that are not included in the files provided by Intel.

After all references to the standard libraries and header files have been changed to the SGX specific ones, the developer will need to setup an ecall for the entry to the application. This allows for the application to be deployed and started from a remote machine. At this point, the application will compile and run but may not be secure to run on an untrusted source. Because all system calls are executed by the OS the application, they will need to be inspected to make sure they do not expose sensitive data without encryption.

Once the application has been inspected and all sensitive data has been encrypted or removed from any system calls, the application will need to be modified to include some kind of exception handling for when Panoply detects malicious or suspicious behavior. The system administrator will want to determine who to notify and if the application should continue to try to execute or just terminate with an error.

After all of these steps have been completed, the application can be deployed and run on the untrusted machine. If the application fails to run as expected, it may be because the underlying OS is not properly running system calls. When this happens, the client should move the application to another machine or cloud provider.

### 4.3 Benefits and Issues

Panoply provides a number of benefits for software developers who want to create or port an application for SGX. It removes a large portion of the initial work which is just getting an application to compile inside of an enclave. Without the Panoply libraries the programmers would have had to redefine most of the C standard data types that their application used, write their own code for any type of input/output operations and remove all system calls from the protected modules of the application. Removing the need to fix all of these basic issues lets the developer focus on the main issue which is ensuring that all sensitive functions are executed inside of a secure enclave.

Another benefit of Panoply is that it allows for developers to separate the sensitive code and data from the portion of the application that does not need to be protected. Other tools which work on compiled binaries such as Graphene do not allow this level of granularity for deciding what does and does not need to be run inside of SGX. The ability to protect only a small segment of an application can improve performance by orders of magnitude. If a large application is running completely inside of an enclave the enclave will constantly be running out of memory. When there is no more space left, the processor has to encrypt and store the data somewhere and then regularly swap it back out and in. Each time the memory needs to be swapped, SGX will have to run the encryption, decryption and hash checks. In addition to the performance issues with swapping memory, it also places the memory in an unprotected environment. Even though the memory is not exposed to the underlying system because it is encrypted, any privileged code could modify it which would prevent the trusted application from running properly. Panoply's granularity allows developers to minimize the size of memory that needs to be protected by SGX, which will improve the speed and robustness of the application.

The convenience that Panoply offers does not come without some risks. Because all system calls are given directly to OS to execute, the likelihood of sensitive data being exposed or improperly modified is high. For large-scale applications there can be thousands of call to system functionality throughout the execution of a program. It is very difficult to check all of the calls and make sure that no unprotected sensitive data is exposed or modified by an untrusted process.

Another concern with using Panoply is that for many applications it will require the modified SDK, which could become a maintanence problem. SGX is being updated by Intel regularly to provide better and more secure features. When these updates occur, they may break the modifications to the SGX driver and SDK. If that happens then the developers will have to either wait for an update to Panoply or fix the modified driver themselves. This just adds another layer of complexity and opens more areas for an error which could cause

data to be compromised or destroyed by a malicious system.

## 4.4 Results

The model that Panoply employs makes it easy to put most any application into an enclave. Panoply doesn't require any major rewrites or additions to get the application up and running, which has opened the door for systems that may have been too complex to modify for SGX without Panoply. The original authors were able to use it to port four medium sized applications using Panoply: Tor, H20, OpenSSL and FreeTDS. Each of these applications has a very large code base but were able to be ported and pass unit and regression tests with less than 1000 lines modified on average [18].

# CHAPTER 5
# MYSQL SECURITY OVERVIEW

MySQL is one of the most popular open-source DBMSs. It is fully scalable and is used by many high profile companies such as Google, Facebook and Twitter. MySQL is often run in cloud environments where the system and data are stored with a public cloud provider and only the small client component, used to query the system, is run in a trusted environment. MySQL supports many security features such as access control systems, SSL connections and data encryption, but these protections are not enough if the operating system becomes compromised or is malicious. This made it a prime candidate for testing the SGX tools. MySQL or any other DBMS will pose problems when porting to SGX because database systems use large amounts of memory to perform operations, and they will have many I/O calls. Porting MySQL is also problematic because it uses hundreds of different system calls in almost all of its modules.

## 5.1 Overview of Database Security Issues and Protections

Database management systems (DBMSs) have become a core part of most major companies information systems. These systems allow for critical information about day to day operations to be stored and tracked without needing to manually find ad-hoc ways to store and index the large sets of data that most companies have today. In recent years, many of these database systems have been moved to the cloud. These cloud based databases make the stored information much more accessible as individuals with appropriate access are able to see and modify the data without needing to have access to the physical storage.

This data stored in these cloud databases is often a critical resource for the organizations, and nearly every major organization is facing problems with database security [19]. The importance of the data makes its protection equally as important. There are many

different mechanisms that can be applied to secure databases against threats, unauthorized users, hackers and IP snoopers.

#### 5.1.1 Data Protections

The primary function of a DBMS is to store information about a subset of the real world and make that information accessible to individuals or groups who need it. Because the information is so important, protecting data from being modified or stolen is one of the primary functions of a database management system.

A DBMS manages access through a layer called an access control system. The purpose of an access control system is to provide protection for all objects stored in a database. The access control system works by managing a matrix of users and objects where the values in the matrix are the permissions that each user has for the associated object. The access control system makes it so each user will only be able to see or modify the data that he or she is authorized to access. Modern database access control systems are effective and efficient [20, 21].

Another protection that DBMS, such as MySQL, can employ to protect the information is encryption. DBMSs may provide options for encrypted individual properties, objects or even the entire data set. Encryption will protect the data in the case that a malicious user gains unauthorized access to the disk where the data is stored. A lot of research has been done on making encrypted databases secure and efficient [22, 23, 24, 25].

#### 5.1.2 Network Protections

Another layer that DBMSs need to protect is the network layer. Users of Cloud based database systems will use a client to establish a connection with the DBMS and then need to authenticate themselves so that they can run queries and commands.

The network layer can be protected by using a secure protocol such as SSL to send and receive data and queries, and using a cryptographic signature scheme to verify that users

are who they claim to be. These two protections along with a good access control system helps ensure that data is not intercepted during transit or sent out to unauthorized users.

#### 5.1.3 Operating System Protections

If the operating system where a DBMS is running is compromised, it could provide a method for an attacker to gain unauthorized access even if the DBMS is otherwise secure. The main protection against such an attack is to keep the OS updated and apply regular security patches.

The problem with OS protection strategies is the requirement that the cloud provider be trusted. Even if the database is encrypted, the access control system is running and all network transmissions are secured, the cloud provider could use privileged code to access the data when it is loaded into memory and could then modify or expose it. To handle this issue, hardware security such as SGX is needed.

### 5.2 MySQL Architecture

MySQL is separated into several components that each perform different tasks. Each of these components can be replaced independently from the others, and the MySQL source code actually includes different implementations that the user can select when building the system.

#### 5.2.1 Client

The first component of the MySQL system is the client. The client application is what the end users will interface with to actually make the initial connection and send commands. The client component will be run from a trusted machine most of the time, so hardware security mechanisms should not be needed.

### 5.2.2 Network Component

The second major piece of the MySQL DBMS is the network component. When a user enters a command in the client, the client connects to the network component to actually send the request to the database. When the network components receives a request it performs several functions. First it authenticates the users credentials to ensure that a user is who they say that they are and that the user has proper access to execute commands.

After a user has been authenticated, the network component receives the users command and then sends it to the next component which is the query processor. In many cases, commands are considered sensitive because they contain information about what the client is doing and may even expose information about data inside the database; however, if the command or query is encrypted by the client before sending it, then there shouldn't be any major security risks for that task.

After the user's command has been execute the network component is also responsible for sending the response back to the client. This could also pose a security concern, but the response can also be encrypted before it is given to this component.

### 5.2.3 Query Processor

The third major component of MySQL is the query processor, which itself has multiple components. The query processor looks at information about the users command and also the underlying structure of the database, so it needs to be protected from snooping, or valuable data could be compromised.

*Query Parser*

The first part of the query processor is the parser. The parser determines what type of action the user is wanting to perform. It figures out what objects will be needed and verifies the user has proper permissions to perform the requested tasks.

*Query Optimizer*

After the query parser has determined what tasks need to be performed, it passes that data to the query optimizer. The optimizer determines exactly how the user's query should be performed. The optimizer then creates an ordered list of operations for MySQL to perform.

### 5.2.4 Storage Engine

The fourth component of MySQL is the storage engine. MySQL uses InnoDB by default but includes several other options and even has an empty model so that developers can write their own. The storage engine is responsible for handling the files and file operations.

In addition to basic file I/O, the storage engine is also responsible for encryption and other tasks related to the logistics of how the data is actually stored on the disk. Because the storage engine component interfaces directly with the data, it needs to be protected when the data is confidential.

### 5.2.5 System Library

The fifth major component of MySQL is the system library called MySys. The MySys library contains basic functions that are used by all of the other components. Example functions in the component include sorting, compression and encryption algorithms. Because these functions are called by all of the other components, the system library would always need to be protected for MySQL to be considered secure.

# CHAPTER 6

# PORTING MYSQL

In this chapter, I present the work that I have done to use SGX and associated technologies. My goal was to port the MySQL database management system (DBMS) to use SGX along with other protections that provide security when running on an untrusted system.

## 6.1 Threat Model

We consider MySQL Server running on a public cloud provider's server with a powerful adversary who has administrative rights to the entire software stack on the server including the hypervisor and operating system. Because of these administrative right, the adversary is able to watch and modify network packets and data for the entire cloud provider's infrastructure. We assume the adversary is not able to physically access or modify the SGX protected memory regions and that all data is encrypted before being written to the disk.

In our threat model, the MySQL client application is running on a trusted machine that the adversary has no access to except for the packets that the client sends. We assume that the client encrypts all packets before transmission, so they will not be exposed when the server loads them into unprotected memory.

## 6.2 Limitations

Using this threat model, an adversary will not be able to view any of the data because it will always be encrypted or protected by SGX. However, an adversary could modify the data, which would prevent MySQL from functioning. In addition, the adversary is able to collect other meaningful data including the size of the data and packets and the amount of traffic and other client information. We consider protecting client and data size information

outside of the scope of this paper.

### 6.3 Porting MySQL with Graphene

I first attempted to port MySQL Server to SGX using Graphene. The first step was to create the required manifest files to setup the basic configurations for running the server. My team and I then created scripts to build and compile all the necessary binaries and tokens and run the test.

Once all of the initial setup was completed, MySQL server was able to compile and run. However, the client was not able to connect to it. After searching for a while, we determined that that issue was caused by a bug in Graphene's LibOS socket bind and we were able to fix it.

After the socket issue was fixed, MySQL was still not running correctly. We determined that Graphene's tests did not provide full code coverage, so there were likely many bugs in the untested functions. In addition to the bugs, Graphene did not implement all of the system calls that MySQL uses. All of these issues showed that even though Graphene has been shown to be an efficient and effective tool for porting applications to SGX, it is not able to support applications as complex as MySQL at this time.

### 6.4 Porting MySQL with Panoply

After attempting to port the entire MySQL server with Graphene did not work, we decided to reduce the scope and try to isolate sensitive components and setup ecalls for them using Panoply.

#### 6.4.1 Security Model

To determine which components needed to be protected inside of an enclave, we first needed to define what data and code was considered sensitive. For simplicity, we let the actual data contained in the database be publicly available. In situations where the data is

public, the sensitive information is what the users do with the data. For public databases, clients will want to protect their access patterns because they may reveal secrets or proprietary information about how they operate.

For this security model, the primary object that needs to be protected is the query itself. To do this, the query would need to be encrypted on transmission, and the query processor and MySys components would need to be run inside of an enclave.

### 6.4.2 Configuring the Build

The first step in porting MySQL to use SGX to protect the queries was to modify the build scripts to use SGX. MySQL is built with CMake scripts whereas all the Panoply and Intel samples use standard makefiles. To overcome this issue, I converted the Intel sample makefile into CMake scripts and then added those scripts into the default MySQL build scripts. A sample of the SGX CMake code is included in Appendix C.

After the build was configured to use the SGX driver and SDK, I then restructured the source code folders to separate the enclave components from the unprotected components. Then, I added the Panoply application and enclave libraries to these new folders. Finally, I setup ecall wrappers to call the protected components inside of the enclave.

### 6.4.3 Errors

After everything was setup, the compilation still failed because Panoply did not define all the functions and types that were needed to run MySQL. A list of definitions needed by MySQL but not implemented or supported by Panoply is included in Appendix B.

To fix this issue, I added the additional functions to the Panoply libraries, but it still wouldn't run because of issues with threads. SGX uses a non standard pthread definition since not all of the default thread functionality is supported inside of an enclave. This created problems because MySQL relies heavily on many of these thread features such as read-write locks. This problem would require a major refactor of the MySQL Server code

to address.

# CHAPTER 7

# RELATED WORK

A large number of organizations are wanting to take advantage of the benefits that cloud providers offer. There has been a lot of research done on how to securely store and process data. Public cloud servers present a new range of security concerns because the users have no access to the physical systems or the processes and users that have elevated privileges for the machine. These problems are currently being addressed through a combination of existing cryptography techniques as well as utilizing secure hardware mechanisms to protect both code and data.

### 7.0.1 Secure Databases

One of the primary candidate systems for using a cloud infrastructure is a database management system. These systems need to store and process large quantities of data and can grow in size rapidly. Because databases are popular tools to run in cloud environments several solutions have been developed to use cryptographic schemes to protect the data without incurring an unreasonable overhead. CryptDB and Arx are examples of databases that encrypt all the data and are able to run queries on the encrypted data [22, 23]. These systems protect the data in the event that the disk is compromised, but cannot protect against an adversary who has access to the unencrypted memory used to process the system's code.

One way to reduce the security risks associated with running any processes on untrusted machines is to combine these database encryption techniques with some form of hardware security. There has been several works to combine encryption in a DBMS with trusted hardware. TrustedDB and Cipherbase are examples of systems that leverage non-SGX hardware security mechanisms to protect queries [26, 25]. While these systems do provide some additional security guarantees, they do not protect the integrity and confidentiality of

all data and code.

#### 7.0.2 SGX

One of the most widely used mechanisms for deploying applications with hardware protections is Intel® SGX. SGX is able protect both code and data in memory from privileged processes. Most of the applications currently using SGX are small and relatively simple, but there is a growing demand to use SGX to support large scale and enterprise level systems with SGX. There have been several projects developed that demonstrate that SGX can be used on larger systems. SecureKeeper is an implementation of Apache ZooKeeper that uses SGX to protect all the user provided data [27]. SecureKeeper keeps the ZooKeeper managed data protected by multiple small enclaves and has an average overhead of 11% over the unmodified ZooKeeper code base. VC3 is a solution for running MapReduce jobs while protecting all the data and code using SGX enclaves [28]. VC3 runs on an unmodified Hadoop but does not expose data to the Hadoop framework. VC3 is able to perform these MapReduce jobs using SGX with an average overhead of around 8%.

In addition to engineering solutions for running specific systems with SGX, several projects have been done to provide mechanisms for porting any application to utilize SGX capabilities. Haven is a tool for running unmodified Windows programs with SGX similar to what Graphene does for Linux. Haven uses a slightly modified LibOS to emulate unsupported calls and features inside of an enclave. Haven was shown to be able to run several larger systems including SQL Server and Apache using SGX; however, it does not guarantee integrity for distributed computations because user-mode scheduling cannot be securely implemented inside of an enclave [29].

Another tool that has been developed to aid in porting applications to use SGX is called SCONE [30]. SCONE works by adding SGX support to Docker containers to support unmodified applications. Docker is tool for container-based virtualization where applications are run in containers which contain the required system functions but do not have the full

overhead of running an application inside of a VM. SCONE provides a secure standard C library interface that includes support for user-level threading and asynchronous function calls inside of enclaves as well as providing extra security features including automatic encryption and decryption for I/O calls. SCONE was demonstrated to work for Apache, Redis, Memcached and NGINX with reasonable performance overheads.

# CHAPTER 8
# CONCLUSION

This report explores the use of Intel SGX for adding security to applications where security was not previously possible. Intel SGX protects application memory at the hardware level, providing security even on cloud systems where the host is not trusted. My goal was to apply the SGX technology to secure the MySQL database management system (DBMS). I used native SGX commands in an attempt to port the MySQL application to SGX. The native SGX commands do not currently have enough capability to port the full MySQL implementation without major refactoring of the MySQL application code.

To overcome the limitations of native SGX, Graphene and Panoply software were employed to assist in porting the MySQL application to SGX. Graphene was designed to import an application with no modifications. However, the technology is still new and multiple bugs prevented the successful deployment of MySQL to SGX using Graphene. Panoply, on the other hand, is designed to support using SGX to protect only the most sensitive components of an application; however, even the subsets of MySQL which were most sensitive needed capabilities not supported by Panoply.

My work has demonstrated that the current SGX environment and associated tools have not yet evolved sufficiently to support complex tasks. The complexity needed for porting MySQL to SGX was more than the current tools could handle, even after making modifications.

## 8.1 Future Work

A possible direction for this work to continue would be to complete and verify a library OS tool such as Graphene. The benefit of a fully implemented and tested library OS tool would be to enable deployment of larger applications such as MySQL to SGX. Once large

software systems are deployed to SGX, metrics could be collected to identify bottlenecks and determine areas where performance improvement is needed. Successful implementation of large applications would enable corporations to move sensitive data to the cloud with protection from untrusted providers.

# Appendices

# APPENDIX A

# SETTING UP SYSTEM CALLS AS OCALLS

This appendix walks through the process of creating an ocall wrapper to call a standard function in the OS that is not supported by SGX. This example uses the basic file operations open, read, write and close defined in `stdio.h`. The first step is to create a new header which includes the missing definitions and place it with the enclave code.

Code A.1: `sgx_stdio.h`

```
#include <stdio.h> // include the original header from the SGX SDK

static inline SGX_FILE *sgx_fopen(const char* filename, const char* mode)
{
    SGX_FILE *f;
    int err = ocall_fopen(f, filename, mode);
    return f;
}
static inline size_t sgx_fread(void* ptr, size_t size, size_t num, SGX_FILE*
    FILESTREAM)
{
    size_t ret;
    ocall_fread(&ret, ptr, size, num, *FILESTREAM);
    return ret;
}
static inline size_t sgx_fwrite(const void* ptr, size_t size, size_t count, SGX_FILE*
    FILESTREAM)
{
    size_t ret;
    ocall_fwrite(&ret, ptr, size, count, *FILESTREAM);
    return ret;
}
static inline int sgx_fclose(SGX_FILE* file)
{
    int ret = 0;
    ocall_fclose(&ret, *file);
    return ret;
}

#define fopen(A, B) sgx_fopen(A, B)
#define fread(A, B, C, D) sgx_fread(A, B, C, D)
#define fwrite(A, B, C, D) sgx_fwrite(A, B, C, D)
#define fclose(A) sgx_fclose(A)
```

After the new header file is created, all references to stdio.h in the application will need to be changed to `sgx_stdio.h`. The next step is to create an edl file to define these ocalls.

Code A.2: `stdio.edl`

```
enclave {
    include "sgx_stdio.h"

    untrusted {
        SGX_FILE *ocall_fopen([in, string] const char* filename, [in, string] const
            char* mode);
        size_t ocall_fread([out, size=size, count=num]void *ptr, size_t size, size_t
            num, SGX_FILE* FILESTREAM);
        size_t ocall_fwrite([in, size=size, count=count]const void * ptr, size_t size,
            size_t count, SGX_FILE* FILESTREAM);
        int ocall_fclose(SGX_FILE* FILESTREAM);
    };
};
```

The final step is to implement the ocalls outside of the enclave.

Code A.3: `stdio.c`

```
#include <stdio.h>

FILE *ocall_fopen(const char* filename, const char* mode)
{
    return fopen(f, filename, mode);
}
size_t ocall_fread(void* ptr, size_t size, size_t num, SGX_FILE* FILESTREAM)
{
    return fread(ptr, size, num, *FILESTREAM);
}
size_t ocall_fwrite(const void* ptr, size_t size, size_t count, SGX_FILE* FILESTREAM)
{
    return fwrite(&ret, ptr, size, count, *FILESTREAM);
}
int ocall_fclose(SGX_FILE* file)
{
    return fclose(&ret, *file);
}
```

## APPENDIX B

## MYSQL REQUIREMENTS NOT SUPPORTED BY PANOPLY

Table B.1: Functions and definitions used by MySQL not supported by Panoply

| File | Function or Definition |
|---|---|
| stdio.h | int sprintf(char * str, const char * format, ...) |
| | FILE * freopen(const char * filename, const char * mode, FILE * stream) |
| string.h | char *stpcpy(char *dest, const char *src) |
| | char *strcpy(char *dest, const char *src) |
| pthread.h | int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock, const pthread_rwlockattr_t *restrict attr) |
| | int pthread_key_create(pthread_key_t *key, void (*destructor)(void*)) |
| | int pthread_key_delete(pthread_key_t key) |
| | int pthread_cond_timedwait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex, const struct timespec *restrict abstime) |
| | int pthread_rwlock_destroy(pthread_rwlock_t *rwlock) |
| | int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock) |
| | int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock) |
| | int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock) |
| | int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock) |
| | int pthread_rwlock_unlock(pthread_rwlock_t *rwlock) |
| unistd.h | int sched_yield() |
| | int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout) |
| | int access(const char *pathname, int mode) |
| signal.h | int kill(pid_t pid, int sig); |
| sys/resource.h | int getrusage(int who, struct rusage *usage) |
| time.h | struct tm *localtime(const time_t *timer) |
| | int clock_gettime(clockid_t clk_id, struct timespec *tp) |
| | CLOCK_REALTIME |
| pwd.h | void endpwent() |
| sys/syscall.h | long syscall(long number, ...) |
| | pid_t gettid() |

# APPENDIX C

# SGX CMAKE SCRIPTS

This appendix illustrates how an SGX application can be built using CMake. All the code is based on the sample Makefiles from the Linux SGX SDK. This appendix includes three sample CMake files one for the root directory, one for the enclave and one for the app. The first code example below (Code C.1) shows a sample file for the root directory of an SGX application.

Code C.1: Sample CMakeLists.txt

```
IF(NOT DEFINED SGX_SDK)
   MESSAGE(STATUS "SGX_SDK is not defined. using /opt/intel/sgxsdk")
   SET(SGX_SDK /opt/intel/sgxsdk)
ENDIF()

IF (NOT DEFINED ENV{SGX_MODE})
   MESSAGE(STATUS "SGX_MODE is not defined. using SIM")
   SET(ENV{SGX_MODE} "SIM")
ENDIF()

IF(NOT DEFINED ENV{SGX_ARCH})
   MESSAGE(STATUS "SGX_ARCH is not defined. using x64")
   SET(ENV{SGX_ARCH} "x64")
ENDIF()

IF(CMAKE_SIZEOF_VOID_P EQUAL 8)
   MESSAGE(STATUS "64 bits compiler detected")
ELSE()
   MESSAGE(STATUS "32 bits compiler detected")
   SET(ENV{SGX_ARCH} "x86")
   MESSAGE(STATUS "SGX_ARCH: x86")
ENDIF()

IF($ENV{SGX_ARCH} MATCHES "x86")
   SET(SGX_COMMON_CFLAGS "-m32")
   SET(SGX_LIBRARY_PATH "${SGX_SDK}/lib")
   SET(SGX_ENCLAVE_SIGNER "${SGX_SDK}/bin/x86/sgx_sign")
   SET(SGX_EDGER8R "${SGX_SDK}/bin/x86/sgx_edger8r")
ELSE()
   SET(SGX_COMMON_CFLAGS "-m64")
   SET(SGX_LIBRARY_PATH "${SGX_SDK}/lib64")
   SET(SGX_ENCLAVE_SIGNER "${SGX_SDK}/bin/x64/sgx_sign")
   SET(SGX_EDGER8R "${SGX_SDK}/bin/x64/sgx_edger8r")
```

```
ENDIF()

IF($ENV{SGX_DEBUG} MATCHES "1")
   IF($ENV{SGX_PRERELEASE} MATCHES "1")
      MESSAGE(FATAL_ERROR "Cannot set SGX_DEBUG and SGX_PRERELEASE at the same time!")
   ENDIF()
ENDIF()

IF($ENV{SGX_DEBUG} MATCHES "1")
   SET(SGX_COMMON_CFLAGS "${SGX_COMMON_CFLAGS} -O0 -g")
ELSEIF(WITH_DEBUG)
   SET(SGX_COMMON_CFLAGS "${SGX_COMMON_CFLAGS} -O0 -g")
ELSE()
   SET(SGX_COMMON_CFLAGS "${SGX_COMMON_CFLAGS} -O2")
ENDIF()

IF($ENV{SGX_MODE} MATCHES "HW")
   MESSAGE(STATUS "Configuring with SGX_MODE= ${SGX_MODE}")
   SET(Urts_Library_Name "sgx_urts")
   SET(Trts_Library_Name "sgx_trts")
   SET(Service_Library_Name "sgx_tservice")
   SET(App_Link_Flags "-lsgx_uae_service_sim")
   SET(Vrf_Cert_File "vrfcert.signed.so_HW")
ELSE()
   SET(Urts_Library_Name "sgx_urts_sim")
   SET(Trts_Library_Name "sgx_trts_sim")
   SET(Service_Library_Name "sgx_tservice_sim")
   SET(App_Link_Flags "-lsgx_uae_service")
   SET(Vrf_Cert_File "vrfcert.signed.so_SIM")
ENDIF()

SET(App_Name "app")

FILE(GLOB App_Cpp_Files "${CMAKE_SOURCE_DIR}/App/App.cpp" "${CMAKE_SOURCE_DIR}/App/
    Edger8rSyntax/*.cpp" "${CMAKE_SOURCE_DIR}/App/TrustedLibrary/*.cpp")
SET(App_Include_Paths "-IInclude -IApp -I${SGX_SDK}/include -I${CMAKE_SOURCE_DIR}/
    include -I${CMAKE_SOURCE_DIR}/Enclave/include")
SET(App_C_Flags "${SGX_COMMON_CFLAGS} -fPIC -Wno-attributes ${App_Include_Paths}")

# Three configuration modes - Debug, prerelease, release
# Debug - Macro DEBUG enabled.
# Prerelease - Macro NDEBUG and EDEBUG enabled.
# Release - Macro NDEBUG enabled.
IF($ENV{SGX_DEBUG} MATCHES "1")
   SET(App_C_Flags "${App_C_Flags} -DDEBUG -UNDEBUG -UEDEBUG")
ELSEIF(WITH_DEBUG)
   SET(App_C_Flags "${App_C_Flags} -DDEBUG -UNDEBUG -UEDEBUG")
ELSEIF($ENV{SGX_PRERELEASE} MATCHES "1")
   SET(App_C_Flags "${App_C_Flags} -DNDEBUG -DEDEBUG -UDEBUG")
ELSE()
   SET(App_C_Flags "${App_C_Flags} -DNDEBUG -UEDEBUG -UDEBUG")
ENDIF()

SET(App_Cpp_Flags "${App_C_Flags} -std=c++11")
```

```
SET(App_Link_Flags "${App_Link_Flags} ${SGX_COMMON_CFLAGS} -L${SGX_LIBRARY_PATH} -l${
    Urts_Library_Name} -lpthread")

SET(Crypto_Library_Name "sgx_tcrypto")

FILE(GLOB Enclave_Cpp_Files "${CMAKE_SOURCE_DIR}/Enclave/Enclave.cpp" "${
    CMAKE_SOURCE_DIR}/Enclave/Edger8rSyntax/*.cpp" "${CMAKE_SOURCE_DIR}/Enclave/
    TrustedLibrary/*.cpp")
SET(Enclave_Include_Paths "-IInclude -I${CMAKE_SOURCE_DIR}/Enclave -I${SGX_SDK}/
    include -I${SGX_SDK}/include/tlibc -I${SGX_SDK}/include/stlport -I${SGX_SDK}/
    include/stlport/using -I${CMAKE_SOURCE_DIR}/Enclave/include")
SET(Enclave_C_Flags "${SGX_COMMON_CFLAGS} -nostdinc -fvisibility=hidden -fpie -fstack-
    protector ${Enclave_Include_Paths}")
SET(Enclave_Cpp_Flags "${Enclave_C_Flags} -std=c++03 -nostdinc++")
SET(Enclave_Link_Flags "${SGX_COMMON_CFLAGS} -Wl,--no-undefined -nostdlib -
    nodefaultlibs -nostartfiles -L${SGX_LIBRARY_PATH} \
   -Wl,--whole-archive -l${Trts_Library_Name} -Wl,--no-whole-archive \
   -Wl,--start-group -lsgx_tstdc -lsgx_tstdcxx -l${Crypto_Library_Name} -l${
       Service_Library_Name} -Wl,--end-group \
   -Wl,-Bstatic -Wl,-Bsymbolic -Wl,--no-undefined \
   -Wl,-pie,-eenclave_entry -Wl,--export-dynamic \
   -Wl,--defsym,__ImageBase=0 \
   -Wl,--version-script=${CMAKE_SOURCE_DIR}/Enclave/Enclave.lds")

SET(Enclave_Name "enclave.so")
SET(Signed_Enclave_Name "Enclave.signed.so")
SET(Signed_Vrfcert_Name "vrfcert.signed.so")
SET(Enclave_Config_File "${CMAKE_SOURCE_DIR}/Enclave/Enclave.config.xml")

IF($ENV{SGX_MODE} MATCHES "HW")
   IF(NOT $ENV{SGX_DEBUG} MATCHES "1")
      IF(NOT $ENV{SGX_PRERELEASE} MATCHES "1")
         SET(Build_Mode "HW_RELEASE")
      ENDIF()
   ENDIF()
ENDIF()

ADD_CUSTOM_TARGET(${Signed_Vrfcert_Name} ALL)

ADD_CUSTOM_COMMAND(TARGET "${Signed_Vrfcert_Name}"
   PRE_BUILD
   COMMAND cp ${CMAKE_SOURCE_DIR}/${Vrf_Cert_File} ./${Signed_Vrfcert_Name}
   COMMAND "echo" 'CP => ${Signed_Vrfcert_Name}'
)

ADD_SUBDIRECTORY(Enclave)
ADD_SUBDIRECTORY(App)

ADD_CUSTOM_TARGET(${Signed_Enclave_Name}
   ALL
   DEPENDS ${Enclave_Name}
)

ADD_CUSTOM_COMMAND(TARGET "${Signed_Enclave_Name}"
```

```
  PRE_BUILD
  COMMAND "${SGX_ENCLAVE_SIGNER}" sign -key "${CMAKE_SOURCE_DIR}/Enclave/
      Enclave_private.pem" -enclave $<TARGET_FILE:${Enclave_Name}> -out "${
      Signed_Enclave_Name}" -config "${Enclave_Config_File}"
  COMMAND echo "'SIGN => ${Signed_Enclave_Name}'"
)

INCLUDE_DIRECTORIES(${SGX_SDK}/include
  ${CMAKE_SOURCE_DIR}/Enclave
  ${CMAKE_SOURCE_DIR}/App
)
```

Code C.2 shows a sample CMake file for the enclave directory of an SGX application

Code C.2: Sample Enclave/CMakeLists.txt

```
SET(CMAKE_C_FLAGS "${Enclave_C_Flags}")
SET(CMAKE_CXX_FLAGS "${Enclave_Cpp_Flags}")

INCLUDE_DIRECTORIES(
   ${CMAKE_SOURCE_DIR}/Enclave/include
   ${SGX_SDK}/include
   ${SGX_SDK}/include/tlibc
)

# EDGER Generated Files
SET(Enclave_T_Files Enclave_t.c Enclave_t.h)
SET_SOURCE_FILES_PROPERTIES(${Enclave_T_Files} PROPERTIES GENERATED TRUE)

# Add all the enclave functions including ecalls/ocalls and syscall wrappers
FILE(GLOB Enclave_Cpp_Files "*.cpp" "TrustedLibrary/*.cpp" "include/*.cpp")
FILE(GLOB Enclave_C_Files "*.c")
FILE(GLOB Enclave_H_Files "*.h" "include/*.h")

# Remove the EDGER generated files from the lists
LIST(REMOVE_ITEM Enclave_C_Files "${CMAKE_CURRENT_SOURCE_DIR}/Enclave_t.c")
LIST(REMOVE_ITEM Enclave_H_Files "${CMAKE_CURRENT_SOURCE_DIR}/Enclave_t.h")

# Generate the trusted files
ADD_CUSTOM_COMMAND(OUTPUT ${Enclave_T_Files}
   COMMAND "${SGX_EDGER8R}" --trusted Enclave.edl --search-path "${CMAKE_SOURCE_DIR}/
       Enclave" --search-path "${SGX_SDK}/include"
   COMMAND cp ${Enclave_T_Files} "${CMAKE_SOURCE_DIR}/Enclave"
   COMMAND "echo" 'GEN => MysqlEclave/Enclave_t.c'
)

ADD_CUSTOM_TARGET(Gen_Enclave_T ALL DEPENDS ${Enclave_T_Files})

# Create enclave libraries
ADD_LIBRARY(Enclave_lib ${Enclave_Cpp_Files} ${Enclave_T_Files} ${Enclave_H_Files})
ADD_DEPENDENCIES(Enclave_lib Gen_Enclave_T)

TARGET_LINK_LIBRARIES(Enclave_lib ${Enclave_Link_Flags})

# create and link the enclave library
ADD_LIBRARY(${Enclave_Name} MODULE ${Enclave_T_Files} ${Enclave_C_Files} ${
    Enclave_Cpp_Files} ${Enclave_H_Files})

ADD_DEPENDENCIES(${Enclave_Name} Gen_Enclave_T Gen_Error)
TARGET_LINK_LIBRARIES(${Enclave_Name} ${Enclave_Link_Flags} )
```

Code C.3 shows a sample CMake file for the app directory of an SGX application

Code C.3: Sample App/CMakeLists.txt

```
SET(CMAKE_C_FLAGS "${App_C_Flags}")
SET(CMAKE_CXX_FLAGS "${App_Cpp_Flags}")

SET(App_U_Files "Enclave_u.c" "Enclave_u.h")
SET_SOURCE_FILES_PROPERTIES(${App_U_Files} PROPERTIES GENERATED TRUE)

INCLUDE_DIRECTORIES(
   ${CMAKE_SOURCE_DIR}/App
   ${CMAKE_SOURCE_DIR}/App/Include
   ${CMAKE_SOURCE_DIR}/Enclave/Include
)

FILE(GLOB App_H_Files "*.h" "Include/*.h" "Untrusted_LocalAttestation/*.h")
FILE(GLOB App_C_Files "*.c" "Untrusted_LocalAttestation/*.c")
FILE(GLOB App_Cpp_Files "*.cpp" "TrustedLibrary/*.cpp" "Untrusted_LocalAttestation/*.
    cpp")

LIST(REMOVE_ITEM App_H_Files "${CMAKE_CURRENT_SOURCE_DIR}/Enclave_u.h")
LIST(REMOVE_ITEM App_C_Files "${CMAKE_CURRENT_SOURCE_DIR}/Enclave_u.c")
LIST(REMOVE_ITEM App_Cpp_Files "${CMAKE_CURRENT_SOURCE_DIR}/App.cpp")

ADD_CUSTOM_COMMAND(OUTPUT ${App_U_Files}
   COMMAND "${SGX_EDGER8R}" --untrusted ${CMAKE_SOURCE_DIR}/Enclave/Enclave.edl --
       search-path "${CMAKE_SOURCE_DIR}/Enclave" --search-path "${SGX_SDK}/include"
   COMMAND cp ${App_U_Files} "${CMAKE_SOURCE_DIR}/App"
   COMMAND "echo" 'GEN => App/Enclave_u.c'
)

ADD_CUSTOM_TARGET(Gen_Enclave_U ALL DEPENDS ${App_U_Files})

ADD_LIBRARY(App_lib ${App_Cpp_Files} ${App_C_Files} ${App_U_Files} ${App_H_Files})
ADD_DEPENDENCIES(App_lib Gen_Enclave_U GenError)

ADD_EXECUTABLE(${App_Name} "App.cpp" "${App_U_Files}" "${App_Cpp_Files}" "${
    App_C_Files}" "${App_H_Files}")
ADD_DEPENDENCIES(${App_Name} Gen_Enclave_U "${Signed_Vrfcert_Name}")

TARGET_LINK_LIBRARIES(${App_Name} ${App_Link_Flags} ${App_Cpp_Flags})
```

## REFERENCES


[1] D. Hyde, "A survey on the security of virtual machines," *Dept. of Comp. Science, Washington Univ. in St. Louis, Tech. Rep*, 2009.

[2] S. Al-Haj, E. Al-Shaer, and H. V. Ramasamy, "Security-aware resource allocation in clouds," in *Services Computing (SCC), 2013 IEEE International Conference on*, IEEE, 2013, pp. 400–407.

[3] E. Caron and J. R. Cornabas, "Improving users' isolation in iaas: Virtual machine placement with security constraints," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, IEEE, 2014, pp. 64–71.

[4] T. T. W. Group *et al.*, "The notorious nine: Cloud computing top threats in 2013," *Cloud Security Alliance*, 2013.

[5] F. Lombardi and R. Di Pietro, "Kvmsec: A security extension for linux kernel virtual machines," in *Proceedings of the 2009 ACM symposium on Applied Computing*, ACM, 2009, pp. 2029–2034.

[6] K. Dahbur, B. Mohammad, and A. B. Tarakji, "A survey of risks, threats and vulnerabilities in cloud computing," in *Proceedings of the 2011 International conference on intelligent semantic Web-services and applications*, ACM, 2011, p. 12.

[7] S. V. K. Kumar and S Padmapriya, "A survey on cloud computing security threats and vulnurabilities," *International Journal Innovative Research in Electrical, Electronics, Instrumentation and control Engineering, ISSN (Online)-2321-2004, Print-2321-5526*, 2014.

[8] C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, "A survey on security issues and solutions at different layers of cloud computing," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 561–592, 2013.

[9] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, p. 13, 2009.

[10] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning, "Managing security of virtual machine images in a cloud environment," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, ACM, 2009, pp. 91–96.

[11] S. Subashini and V Kavitha, “A survey on security issues in service delivery models of cloud computing,” *Journal of network and computer applications*, vol. 34, no. 1, pp. 1–11, 2011.

[12] G. S. Bindra, P. K. Singh, K. K. Kandwal, and S. Khanna, “Cloud security: Analysis and risk management of vm images,” in *Information and Automation (ICIA), 2012 International Conference on*, IEEE, 2012, pp. 646–651.

[13] A. Jasti, P. Shah, R. Nagaraj, and R. Pendse, “Security in multi-tenancy cloud,” in *Security Technology (ICCST), 2010 IEEE International Carnahan Conference on*, IEEE, 2010, pp. 35–41.

[14] F. Sabahi, “Cloud computing security threats and responses,” in *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, IEEE, 2011, pp. 245–249.

[15] T. Mandt, M. Solnik, and D. Wang, “Demystifying the secure enclave processor,”

[16] V. Costan and S. Devadas, “Intel sgx explained.,” *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[17] C.-C. Tsai, D. E. Porter, and M. Vij, “Graphene-sgx: A practical library os for unmodified applications on sgx,” in *2017 USENIX Annual Technical Conference (USENIX ATC)*, 2017.

[18] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, “Panoply: Low-tcb linux applications with sgx enclaves,” in *Proc. of the Annual Network and Distributed System Security Symp.(NDSS)*, 2017.

[19] S. P. Singh, A. Singh, and U. N. Tripathi, “Enforcing database security in un-trusted environment by using multisession and biometrics based authentication,” *International Journal of Emergining Research in Management & Technology*, vol. 4, no. 5, 2015.

[20] S. Martínez, V. Cosentino, J. Cabot, and F. Cuppens, “Reverse engineering of database security policies,” in *Database and Expert Systems Applications*, Springer, 2013, pp. 442–449.

[21] R. Gangwar and M. Sharma, “Database security measurements issues in adhoc network,” *International Conference of Advance Research and Innovation (ICARI)*, 2015.

[22] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, “Cryptdb: Protecting confidentiality with encrypted query processing,” in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ACM, 2011, pp. 85–100.

[23] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system.," *IACR Cryptology ePrint Archive*, vol. 2016, p. 591, 2016.

[24] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with cipherbase.," in *CIDR*, Citeseer, 2013.

[25] A. Arasu, K. Eguro, M. Joglekar, R. Kaushik, D. Kossmann, and R. Ramamurthy, "Transaction processing on confidential data using cipherbase," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, IEEE, 2015, pp. 435–446.

[26] S. Bajaj and R. Sion, "Trusteddb: A trusted hardware-based database with privacy and data confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 752–765, 2014.

[27] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, "Securekeeper: Confidential zookeeper using intel sgx," in *Proceedings of the 16th Annual Middleware Conference (Middleware)*, 2016.

[28] 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 *Security and Privacy (SP), 2015 IEEE Symposium on*, IEEE, 2015, pp. 38–54.

[29] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, p. 8, 2015.

[30] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, *et al.*, "Scone: Secure linux containers with intel sgx," in *12th USENIX Symp. Operating Systems Design and Implementation*, 2016.