# Protecting User Data from Adversarial Servers and Clouds

Byoungyoung Lee
*byoungyoung@purdue.edu*

## Abstract

Computation outsourcing to remote clouds is popular today. In particular, enterprises offload various computations to cloud machines, including security critical machine learning workloads, to avoid costly and time-consuming maintenance efforts. Such computation offloading raises many critical security concerns, primarily because data is now being handled by potentially adversarial clouds. Intel SGX is one of the promising hardware-based trusted computing solutions that has recently become available, and has strong potential to address aforementioned problems. Running a program within a protective region, called an enclave, SGX guarantees confidentiality and integrity of the program against all hardware and software components apart from the CPU itself.

However, we observe that SGX alone cannot guarantee the confidentiality of user data, especially if the user who provides input and the program owner are two separate entities. More precisely, SGX only provides primitive security guarantees on confidentiality and integrity of the program, and it does not restrict how the program handles the data provided to it. Missing this security guarantee severely endangers the user's privacy, because the program owner can easily write their program to collect user's data. This proposal proposes SECUREBOOTH, a system that provides confidentiality on user data. SECUREBOOTH creates a software-enforced sandbox for a program running within an enclave. SECUREBOOTH further ensures that all outgoing data transmissions from the program are encrypted in such a way that it is only visible to the corresponding user. SECUREBOOTH designs software fault isolation (SFI) technique for SGX, to secure each thread from others in an SGX environment. We plan to implement SECUREBOOTH based on real SGX hardware, and evaluate it with real-world applications such as Hadoop, MySQL databases and Apache web servers.

## Problem Statement and Motivation

In the past several years, hardware-based trusted computing solutions have become a reality. Intel SGX is one such solution that has recently become commoditized with the release of Intel Skylake processors. Intel SGX guarantees confidentiality and integrity of a program without trusting the underlying software components including kernel, hypervisor, and most hardware components except the CPU itself. One of the most attractive security applications of SGX would be offloading computational workloads to untrusted or potentially adversarial clouds. Intel SGX supports remote attestation on the program to be loaded on the cloud and further ensures that the runtime execution context of the program is not visible to any entity (including privileged cloud hosts) outside its protective region, called an enclave.

However, we observe that Intel SGX alone cannot provide complete security, if the program (to be run on clouds) and the data (to be processed by the program) are provided by two different parties. This is a common scenario embodied in the client-server model where the server runs the program on the cloud and the client hands over the data to the server. The server wishes to run its program as intended on the cloud and provide a service for users. At the same time, the server might wish to collect user data, as data itself can be monetized in many different ways (i.e., personalized advertising, market research, etc.). In this setting, running native Intel SGX will protect the interests of both parties against the adversarial clouds (or any other privileged attacker), but will fail to prevent an adversarial server from collecting user data. This is largely because Intel SGX considers both parties as the same entity from a security standpoint and therefore enforces the same security privilege for both parties.

As a result, an adversarial server can write its program such that it intentionally leaks the client's data, since Intel SGX does not perform checks on the runtime behavior of a program. A client, wary of such an attack, might attempt to inspect the server's program beforehand to check if it leaks data, but manually or formally verifying such a property is a challenge in its own. Even worse, a server may not provide its implementation at all because of proprietary concerns, which simply forbids inspection by a client. Based on the current trends, we find the severity of such issues is amplified because of the sheer amount of personal data being circulated and processed by untrusted entities. Consider companies like 23andMe or AncestryDNA, which provide DNA testing services and therefore, handle and process privacy-critical DNA samples provided by users. Envisioning the future uses of SGX, database services can be equipped with the notion of private information retrieval (PIR), which can protect the confidentiality of the query that the client sent. As another example, companies want to setup Intrusion Detection Systems (IDS) with deep packet inspection (DPI) capabilities, but at the same time guaranteeing the confidentiality of user's network traffic. All of these services mentioned above might promise in their service agreement that user data would not be collected (or propagated), but there is no existing way to ascertain such a promise.

## Approach

This proposal proposes SECUREBOOTH, a system that addresses this security conundrum in SGX environments. SECUREBOOTH aims at protecting the confidentiality of a user's data against an adversarial server. Towards this end, SECUREBOOTH guarantees following security properties to safeguard the confidentiality of user data from adversarial clouds: **(P1) Server isolation**: SECUREBOOTH should deny server writes to the outside of the enclave in order to prevent it from leaking the user data; **(P2) Thread isolation**: SECUREBOOTH should isolate each thread of the server from others to avoid information leakage between threads, as each thread runs for a different client; **(P3) Encrypt outgoing user data**: SECUREBOOTH always has to encrypt all user's data (including data having dependencies with the user's data), when it leaves the enclave. This encryption has to be performed using a session key such that only the intended user can decrypt; and **(P4) Complete Interaction Mediation**: SECUREBOOTH has to completely mediate all types of interactions originating from a target server program to stop potential side-channels from leaking information.

To ensure aforementioned four properties, we present a sandboxed execution environment, which runs inside an enclave using the Software Fault Isolation (SFI) technique. By leveraging SFI, SECUREBOOTH limits the memory access range of the server with the following two rules. First, it does not allow the server to directly write data outside the enclave (P1). This prevents the server running within the enclave from disclosing enclave data (i.e., user private data). Second, SECUREBOOTH provides a thread of the server with its own private region and it prohibits a thread from reading/writing data to the memory region owned by another thread that runs in the same enclave (P2). This prevents a thread from leaking the user private data handled by another thread. In addition, SECUREBOOTH provides a limited set of runtime interfaces to allow the server to securely transmit service results or client data to the outside world. The transmitted data is encrypted with the corresponding client's session key so that only the client can decrypt it (P3). Also, SECUREBOOTH mediates all interactions originating from the server through the runtime interface to prevent potential side-channel attacks (P4).

We plan to implement SECUREBOOTH using LLVM and systematically integrate it with the Graphene Library Operating system to demonstrate its practicality as far as running real-world applications is concerned. In addition, SECUREBOOTH plans to adopt the following three components as well: (1) SSL/TLS to establish secure channels with both server and users, (2) dynamic loader/linker to bootstrap enclave programs, and (3) disassembler validating the permission and security enforcement. We will thoroughly evaluate SECUREBOOTH using real SGX hardware (i.e. Intel Skylake) to understand the impact on security as well as performance under such a system.

## Plan

We expect SECUREBOOTH can be implemented and evaluated within an year. **Months 1-4**: The student will develop LLVM-based software fault isolation techniques for SECUREBOOTH. This phase involves two tasks: clearly identifying unique memory layouts imposed by SGX and designing efficient instrumentation techniques for SGX. **Month 5-8**: The student will develop runtime engines for SECUREBOOTH. Integrating SECUREBOOTH's software fault isolation techniques, this runtime engine performs secure encryption of outgoing user's data. **Month 9-12**: In this phase, the student will evaluate and test SECUREBOOTH. We target real-world applications in the end, including Hadoop, MySQL databases, and Apache web servers.

## Budget

PI plans to support one PhD student on this project for one year. The requested budget covers the student's tuition, salary, and graduate fee remissions. The total estimated budget is $75,101.