General and secure support of legacy computations on TEE

Yongzhi Wang
Yongzhi.Wang@tamucc.edu
Department of Computer Science, *Texas A&M University-Corpus Christi*

Introduction

The emergence of Trusted Execution Environments (TEE) provides a promising path to protect the confidentiality of remote computing. How to use TEE to protect legacy applications is a hot topic. Until now, a wide range of solutions have been proposed to protect a specific class of computations on a specific TEE environment: Some works ported a specific application to TEE (e.g., EnclaveDB on Intel SGX). Some works ported a language interpreter or language library to TEE to support the applications written in that language (e.g., RUST-SGX, Python-SGX, GoTEE, Civet, etc.). Some works supported more general binary executions by porting library OS or C library to TEE (e.g., SCONE, Graphene-SGX, SGX-LKL, etc.). All those works have two major limitations: 1) Moving applications, language interpreters or libraries to TEE leads to large trusting base, which causes security loopholes; 2) Each work focused on a specific (class of) application(s), which cannot be transferred to other applications, languages, or libraries.

In this work, we introduce a language neutral solution, namely EnCloak, which can support multiple programming languages while maintaining small trusting base. To achieve this goal, our solution identifies sensitive statements in the source code, translates them into the proposed language neutral, platform neutral Enclave Instructions(EIs) and executes them in the TEE, called Cloak Enclave. To execute the EIs in TEE, we implement the EI run-time system in Cloak Enclave, which manages sensitive variables involved in the sensitive statements and executes the translated EIs. The architecture of EnCloak is shown in Fig. 1.

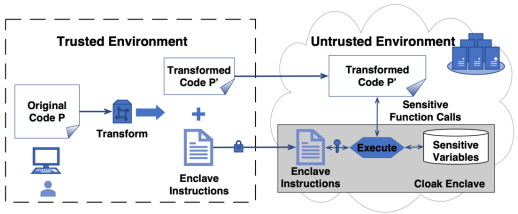


Figure 1: The EnCloak Architecture

System Design

To use EnCloak to protect sensitive data in a remote computing, the user first tags sensitive sources in the source code. The EnCloak then performs taint analysis to identify all sensitive variables. After that, EnCloak replaces each statement s containing sensitive variables with a designed function call that will be executed in Cloak Enclave. EnCloak supports two types of sensitive function calls: update and evaluate.

$$update(is, uuid, ouuid) \quad (1)$$

$$evaluate(is, uuid, ouuid) \quad (2)$$

The former will update sensitive variables in the Cloak Enclave based on the variables outside of TEE. The later is for the code outside of TEE to evaluate a branch condition based on the variable values inside of TEE. In the above function calls, is is an identifier of statement s. $uuid$ uniquely identifies a function execution of the transformed program. It helps locating sensitive variables managed in the Cloak Enclave. For the same function that are executed two times, the two executions will have distinct $uuid$ to distinguish each other.

To manage sensitive variables and support the execution of EIs, we design a runtime environment in the Cloak Enclave. In the runtime environment, sensitive variables from different function executions or objects are properly isolated and managed, as shown in Fig. 2.

The format of EIs are extended from three address code. An example of EI can be

$$is :< ASSIGN, \diamond, left, right, dest > \quad (3)$$

, where is is the reference index that is used to match is in the function call listed in (1) and (2). We extend the format of EI to support complex features including inter-procedure function invocation, multi-dimensional arrays, etc.

Generality

We have implemented a prototype system based on the design of EnCloak, which support Java programs running on Intel SGX. The design of EnCloak can be transferred to other languages and TEE platforms. With the language and platform neutral EI, to support computations written in a specific language on a specific TEE, developers only need to write a parser to translate the language to EI and implement a runtime to support EI on that TEE.

Security

EnCloak treats security as the first-class citizen. Since it only moves sensitive statements to TEE, it significantly reduces the trusting base. Additionally, the execution of EIs and access of sensitive variables can be designed to satisfy different security goals, e.g., memory access obliviousness.

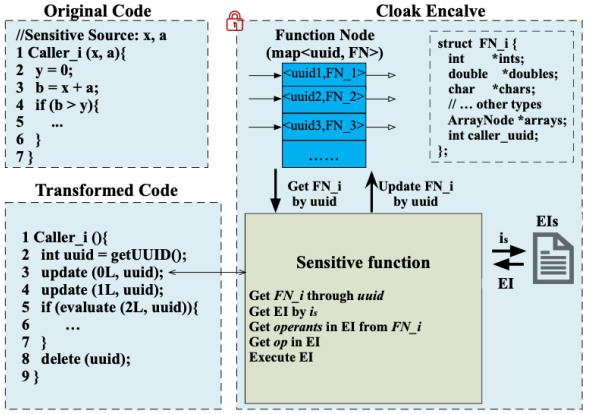
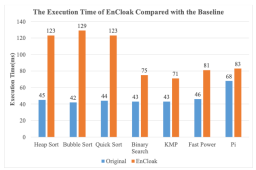
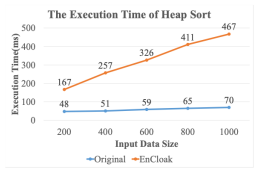


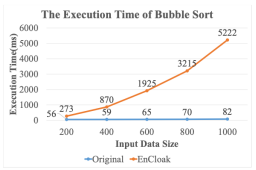
Figure 2: Function Node in Cloak Enclave



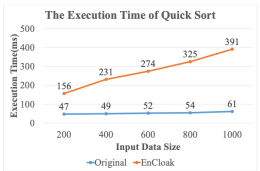
(a) Execution times



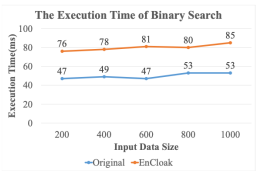
(b) Heap Sort



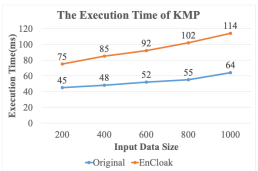
(c) Bubble Sort



(d) Quick Sort



(e) Binary Search



(f) KMP

CPU intensive applications. (a) shows the execution time of different applications, (b) - (f) show the scalability results.