OpenEnclave Design Overview

September 4, 2017

1. Introduction

This document provides a brief design overview of the **OpenEnclave SDK** (**OE**). It describes the parts of the SDK and how they work together to create, invoke, and terminate enclaves. This document assumes the reader knows what an enclave is.

OE aims to provide an abstract API for developing enclave applications whether they are based on the Intel® Software Guard Extensions (SGX) or the Microsoft® Virtual Secure Mode (VSM). This early version though only supports SGX, so the design discussion that follows refers exclusively to SGX.

2. The Enclave Application

OE helps developers build **enclave applications**. An enclave application is partitioned into an untrusted component (called a **host**) and a trusted component (called an **enclave**). An enclave is a secure container whose memory (text and data) is protected from access by outside entities, including the host, privileged users, and even the hardware.

On Linux, enclaves are packaged as shared objects that have been digitally signed. For example, an enclave called **turtle** might be redistributed with the following filename.

```
turtle.signed.so
```

A host may load and instantiate this enclave as shown by the following snippet.

```
OE_Enclave* enclave;
OE_Create("turtle.signed.so", 0, &enclave);
```

The **enclave** variable is a handle used to refer to the enclave throughout its lifetime. A host may wish to create several enclaves, either of the same type or different types.

Once an enclave is created, the host may invoke its functions, known as **enclave calls** or **ECALLs**. In the snippet below, the host calls the enclave's **Walk** function.

```
OE_CallEnclave(enclave, "Walk", args));
```

This enters the enclave and calls its Walk function. To service this request, the turtle enclave implements a Walk function with the following signature.

```
OE_ECALL void Walk(void* args);
```

Once an enclave is entered, the enclave may call functions within the host, called **outside calls** or **OCALLs**. For example, the enclave may call the host's **WhoAreYou** function as shown in the snippet below.

```
OE_CallHost(enclave, "WhoAreYou", args));
```

To service this request, the host implements a WhoAreYou function with the following signature.

```
OE_OCALL void WhoAreYou(void* args);
```

The **args** parameter for ECALLs and OCALLs is defined by the developer of the enclave application. It might be a pointer to a string or a C structure. Although **OE_CallEnclave** and **OE_CallHost** are not type safe, we will see later how to use the IDL stub generator (**oegen**) to produce type-safe, parameterized stubs for safely calling them.

The host and enclave may continue to exchange ECALLs and OCALLs. ECALLs and OCALLs may be arbitrarily nested as stack memory allows.

Eventually the host will terminate the enclave as shown in the snippet below.

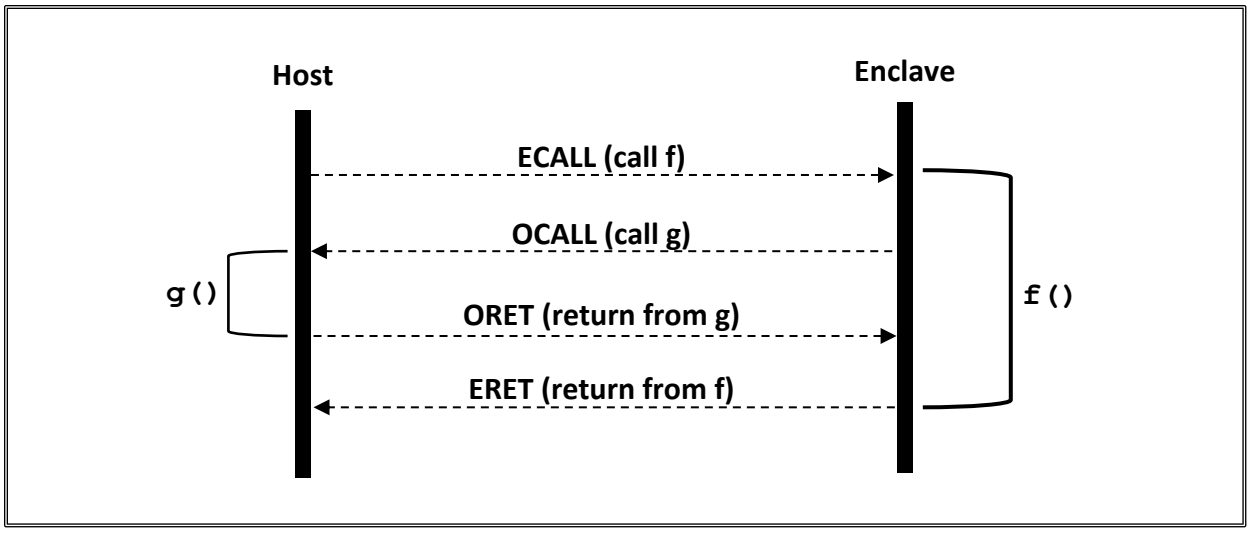
```
OE_TerminateEnclave(enclave);
```

3. The Inter-Call Model

OE defines an **inter-call model** whereby the host and enclave call each other's functions. These may be thought of as messages exchanged between the host and enclave. There are four kinds of messages, defined in the table below.

Message	Description	Sender	Receiver
ECALL	An enclave call	Host	Enclave
ERET	An enclave return	Enclave	Host
OCALL	An outside call	Enclave	Host
ORET	An outside return	Host	Enclave

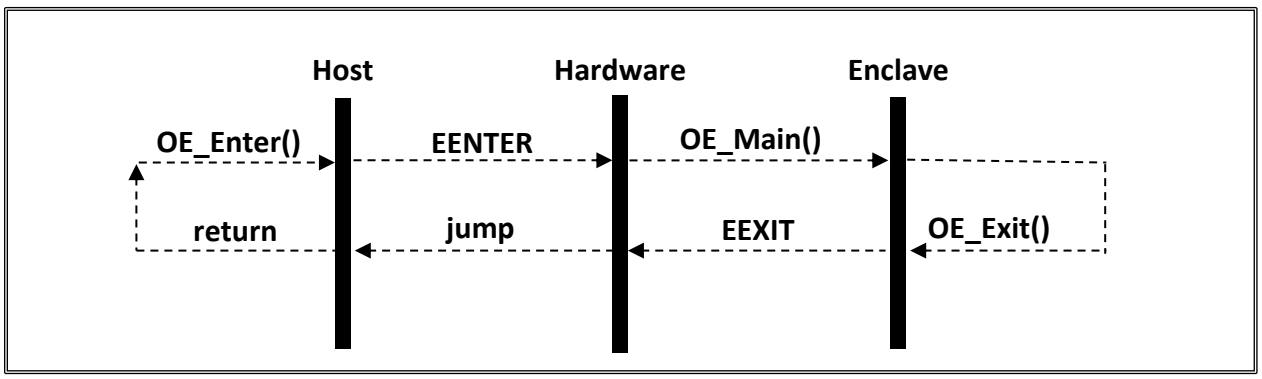
An enclave may perform an OCALL while servicing an ECALL, and a host can perform an ECALL while servicing an OCALL. So ECALLs and OCALLs may be nested arbitrarily. The diagram below illustrates one level of nesting.



ECALLs and OCALLs (and their respective returns) are software constructs. There is no direct analogue with SGX hardware instructions. Rather these calls are implemented using the **EENTER** and **EEXIT** instructions as defined in the table below.

Instruction	Description
EENTER	Executed by the host (**OE_Enter**) to enter the enclave (**OE_Main**)
EEXIT	Execute by the enclave (**OE_Exit**) to return to host (**OE_Enter**)

The interaction between the host and enclave during execution of these instruction is depicted below along with the OE functions that are involved.



ECALLs and OCALLs are layered on the EENTER and EEXIT instructions. The table below shows how the four message types are mapped onto the SGX instructions.

Message	SGX Instruction	Explanation
ECALL	**EENTER**	ECALL executes EENTER to enter enclave
ERET	**EEXIT**	ERET executes an EEXIT to exit enclave
OCALL	**EEXIT**	OCALL executes EEXIT to exit enclave
ORET	**EENTER**	ORET executes EENTER to reenter the enclave

When the host performs an ECALL, it executes an EENTER instruction to enter the enclave. The host blocks until the EENTER instruction returns (that is when the enclave executes EEXIT). The enclave exits to perform an ERET or an OCALL. The reason for the exit is returned in a register. The host examines this register to determine what to do next: to process an ERET or to service an OCALL.

The table below identifies key OE functions that participate in ECALLs and OCALLs along with source code references for each.

Function	Description	Source Location
OE_Enter()	Host calls to execute EENTER to enter an enclave.	host/enter.S
OE_Main()	EENTER calls this enclave entry point.	enclave/main.S
OE_Exit()	Enclave calls to execute EEXIT to exit the enclave.	enclave/exit.S

These behavior of these functions is defined below.

OE_Enter()

OE_Enter() executes the EEXIT instruction with the following register assignments.

- RDI: The address of a Thread Control Structure (TCS) in the enclave
- RSI: Address of an Asynchronous Exception Procedure in the host
- RDX:
    - High-word: code indicating whether ECALL or ORET
    - Low-word: function number
- RCX: ECALL or ORET argument

EEXIT obtains the enclave entry point from the TCS, which is the OE_Main() function. EEXIT calls OE_Main(), passing it the return address (the instruction in the host immediately following the

EENTER instruction execution). The calling thread executes in the enclave until the enclave calls OE_Exit(), which executes the EEXIT instruction, which jumps to the return address in the host.

OE_Main()

The EEXIT instruction calls OE_Main() to enter the enclave with the following register assignments.

- RAX: index of current SSA (non-zero indicates an exception)
- RBX: address of the TCS
- RCX: return address (address of instruction in host immediately following EENTER)
- RDI:
    - High-word: code indicating whether ECALL or ORET
    - Low-word: function number
- RSI: ECALL or ORET argument

OE_Main() performs the following tasks:

- Saves the hosts registers
- Initializes the enclave stack frame
- Invokes __OE_HandleMain()

__OE_HandleMain() handles either an ECALL or an ORET, dispatching it as necessary. __OE_HandleMain() never returns but instead calls OE_Exit() which executes an EEXIT instruction.

OE_Exit()

OE_Exit() is called to either perform an ERET or an OCALL. It performs the following:

- Clears the enclave registers
- Restores the host registers
- Executes the EEXIT instruction with the following register assignment:
    - RBX – return address
    - RCX – Asynchronous Exception Procedure (AEP)
    - RDI:
        - High-word: code indicating whether OCALL or ERET
        - Low-word: function number
    - RSI: OCALL or ERET argument

The host examines the high-word of RDI (the code) to determine what action to take and the low-word of RDI (the function number) to determine what function to call. The function argument is taken from RSI.

4. Enclave Creation