DSP Lab 2016-17, JNI using Intel-SGX

Clindo Devassy K, Subhadeep Manna

Abstract
Software applications frequently need to work with private information. The security of these information is crucial. Intel have introduced the Intel® Software Guard Extensions (Intel® SGX). A method to secure our code and data from disclosure. Intel SGX could be coded using C/C++. This project creates a JNI implementation for SGX calls and with those implementation creates an arithmetic evaluator. This arithmetic evaluator can be accessed remotely.

Keywords
Intel SGX, Java JNI, Arithmetic evaluator

Contents

Introduction

Arithmetic and Logic evaluators for expressions are the backbone of several stand alone and distributed software applications. It can be used for evaluating essential and sensitive expressions for scientific, engineering, banking and many other such platforms. This expression is divided into tokens using tokenizer and the evaluator is used to evaluate the value of each token internally. Finally giving us the final output. We are implementing an arithmetic evaluator using Intel SGX and the SGX is invoked using JNI

1. Enclave Code

When the application that uses this parser highly depends on the genuine and precise result of the expression. It is highly important that the parser is secure from attacks and tampering. So, by using Intel SGX we plan to reduce the attack surface on the arithmetic and logical parser which would be used by the application.

The untrusted application component here is the application presentation layer. Which would be the GUI in this case.

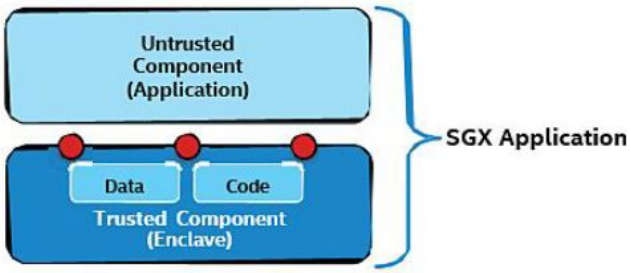

Figure 1. Trusted and untrusted part of an SGX application.

Source: https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation

The trusted component or the enclave code would be the core parser module. This would be written under the ECALL section of an intel SGX enabled application.

The untrusted component would be the user interface and other related module. This would be written under OCALL. <Enclave_Name>.edl file would be used to define the trusted part of the application which is the arithmetic and logic parser module. Below is the enclave code that we plan to implement in our project:

```
// Enclave code
enable_enclave()
{
enclave {


    trusted {
       /* defining ECALLs here*/

   //Input functions to Java application

   //for the arithmetic expression input
       module
```

```
    arithmetic_evaluator_moduleJNICall();

    //for the logical expression input module
    logical_evaluator_moduleJNICall();

        };


    untrusted {
              /* defining OCALLs here */
    //call to the application user interface
   application_userinterfaceJNIcall();

         };
    }
}
```

2. JNI code (and C non-enclave code)

The most fundamental part of our project is to create a SGX enabled application using JNI in java. Two possible ways to approach this:

1. Link the Java application to another SGX enabled C/C++ application. (The disadvantage being performance issue and programmer efforts).
2. The second way to approach the problem is to call just the SGX implemented function written in native C/C++ through JNI in java. This overcomes the problems faced in the first technique.

The steps how we plan to achieve this is as follows:

1. Java module and class declaring the native methods loadLibrary(String path) loads an external C library. The variable path is the path to the library.

```
public class JNI_ImplSGX {

   static {

      // loading the C-library
      System.loadLibrary("C_Encalve_Module");
   }

   // declaration of native method
   private native void enableSGX();

   public static void main(String[]
       args) {
      new JNI_ImplSGX ().enableSGX ();
   }
}
```

2. Generate a C-header file (.h) which has the native function declaration using javah tool.

```
javah -jni JNI_ImplSGX
```

3. Header which is generated contains a declaration of C function linked with the Java native method:

```
JNIEXPORT void JNICALL
Java_enableSGX_call (JNIEnv *, jobject);
```

4. The C code

```
#include <jni.h>
#include <stdio.h>
#include " JNI_ImplSGX.h"

JNIEXPORT void JNICALL
Java_enableSGX_call (JNIEnv *env,
    jobject obj)
 {
enable_enclave(); //enclave call to the
    java application modules
 }
```

5. The above can be compiled into a (dynamic link libraries) .dll file and can be used from the Java application by copying the .dll file generated to the java application and specifying the path in

```
System.Loadlibrary(String path)
```

3. Phases

We are using a Client-Server model for our application with following assumptions

1. Both machines must be SGX enabled.
2. They communicate through via Web Sockets

3.1 Initial setup

The client app and server app must be started on respective hosts as separate SGX enabled applications.

3.2 Initial key exchange

We plan to check the availability of server and client on the network initially by public key exchanges matched against their private keys.

3.3 Authentication client - server

We plan to carry out the authentication of client and server using SGX Remote attestation.

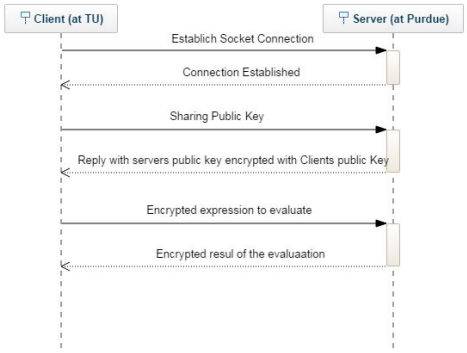

Figure 2. Client Server Authentication Sequence Diagram

3.4 Communication client - server enclave

Based on the assumption that the Client and Server both are running on SGX enabled machine

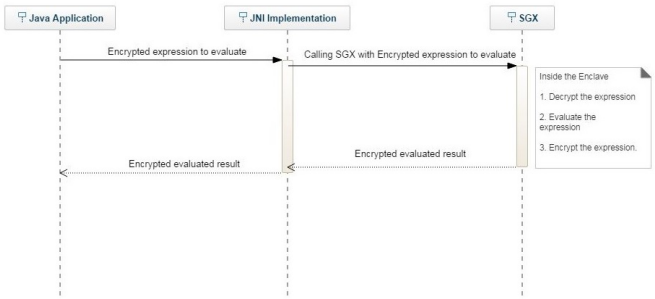

Figure 3. Sequence diagram-Inside SGX

4. Interface / API of the interpreter

The main section of Arithmetic interpreter API is as follows:

1. Parser_Arithmetic : This class is used to take an input string expression. The method parse() is used to pass it to the tokenizer.
2. Core_Tokenizer: It splits the stream into tokens which is then returned to the parser. The tokens are defined in a list class called as Tokens.
3. Exp_Node: This class gets the type of terminal or non terminal symbol. Then passed it to the respective class(to which the token belongs) for the evaluation of the expression.
4. The parser Arithmetic class returns the value of the expression as the Output Value

References

[1] https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation.

[2] https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-exampleintroduction