Intel® Software Guard Extensions Evaluation SDK for Windows* OS

User's Guide

Revision: 1.1.1

World Wide Web: http://www.intel.com

Intel Confidential

Legal Information

Optimization Notice

Optimization Notice
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Revision History

Revision Number	Description	Revision Date
1.1	SGX Win 1.1 release	September 2015
1.1.1	SGX Win 1.1.1 release	January 2016

Introduction

Intel provides the Intel® Software Guard Extensions (Intel® SGX) Evaluation SDK User's Guide for software developers who wish to harden their application's security using Intel Software Guard Extensions technology.

This document covers an overview of the technology, tutorials, tools, API reference as well as sample code.

Intel® Software Guard Extensions Evaluation SDK from Intel is a collection of APIs, sample source code, libraries and tools that enables the software developer to write and debug Intel® Software Guard Extensions applications in C/C++.

NOTE

Intel® Software Guard Extensions(Intel® SGX) technology is currently available only on 6th Generation Intel® Core™ Processor (codenamed Skylake).

NOTE

This document refers to the evaluation software – Intel® Software Guard Extensions Evaluation SDK, which is currently in the design phase and has not been fully validated. Statements regarding functionality, security or performance are therefore subject to change. Additionally, there may be descriptions for possible deployment options of the software. At this stage, these are just ideas. Intel reserves the rights to deploy or use this software in different ways including the option of not deploying a final version of the software.

Intel® Software Guard Extensions Technology Overview

Intel® Software Guard Extensions is an Intel technology whose objective is to enable a high-level protection of secrets. It operates by allocating hardware-protected memory where code and data reside. The protected memory area is called an enclave. Data within the enclave memory can only be accessed by the code that also resides within the enclave memory space. Enclave code can be invoked via special instructions. An enclave can be built and loaded as a Windows* DLL.

NOTE:

The enclave file can be disassembled, so the algorithms used by the enclave developer will not remain secret.

Intel® Software Guard Extensions technology has a hard limit on the protected memory size, typically 64 MB or 128 MB. As a result, the number of active enclaves (in memory) is limited. Depending on the memory footprint of each enclave, use cases suggest that 5-20 enclaves can reside in memory simultaneously.

Intel® Software Guard Extensions Security Properties

- Intel designs the Intel® Software Guard Extensions to protect against software attacks:
  - The enclave memory cannot be read or written from outside the enclave regardless of current privilege level and CPU mode (ring3/user-mode, ring0/kernel-mode, SMM, VMM, or another enclave). The abort page is returned in such conditions.
  - An enclave can be created with a debug attribute that allows a special debugger (Intel® Software Guard Extensions debugger) to view its content like a standard debugger. Production enclaves (non-debug) cannot be debugged by software or hardware debuggers.
  - The enclave environment cannot be entered via classic function calls, jumps, register manipulation or stack manipulation. The only way to call an enclave function is via a new instruction that performs several protect checks. Classic function calls initiated by enclave code to functions inside the enclave are allowed.
  - CPU mode can only be 32 or 64 bit when executing enclave code. Other CPU modes are not supported. An exception is raised in such conditions.
- Intel designs the Intel® Software Guard Extensions to protect against known hardware attacks:
  - The enclave memory is encrypted using industry-standard encryption algorithms with replay protection.
  - Tapping the memory or connecting the DRAM modules to another system will only give access to encrypted data.
  - The memory encryption key changes every power cycle randomly (for example, boot/sleep/hibernate). The key is stored within the CPU and it is not accessible.
  - Intel® Software Guard Extensions is not designed to handle side channel attacks or reverse engineering. It is up to the Intel® SGX developers to build enclaves that are protected against these types of attack.

Intel® Software Guard Extensions uses strong industry-standard algorithms for signing enclaves. The signature of an enclave characterizes the content and the layout of the enclave at build time. If the enclave’s content and layout are not correct per the signature, then the enclave will fail to be initialized and, hence, will not be executed. If an enclave is initialized, it should be identical to the original enclave and will not be modified at runtime.

Application Design Considerations

An Intel® Software Guard Extensions application design is different from non- Intel® SGX application as it requires dividing the application into two logical components:

- Trusted component. The code that accesses the secret resides here. This component is also called an enclave. More than one enclave can exist in an application.
- Untrusted component. The rest of the application including all its modules.[1]

The application writer should make the trusted part as small as possible. It is suggested that enclave functionality should be limited to operate on the secret data. A large enclave statistically has more bugs and (user created) security holes than a small enclave.

[1]From an enclave standpoint, the operating system and VMM are not trusted components, either.

The enclave code can leave the protected memory region and call functions in the untrusted zone (by a special instruction). Reducing the enclave dependency on untrusted code will also strengthen its protection against possible attacks.

Embracing the above design considerations will improve protection as the attack surface is minimized.

The application designer, as the first step to harnessing Intel® Software Guard Extensions Evaluation SDK in the application, must redesign or refactor the application to fit these guidelines. This is accomplished by isolating the code module(s) that access any secrets and then moving these modules to a separate package/library. The details of how to create such an enclave are detailed in the tutorials section. You can also see the demonstrations on creating an enclave in the sample code that are shipped with the Intel® Software Guard Extensions Evaluation SDK.

Terminology and Acronyms

AE	Architectural enclaves. Enclaves that are part of the Intel® Software Guard Extensions framework. They include the quoting enclave (QE), provisioning enclave (PvE), launch enclave (LE) and the platform service enclave (PSE).
Attestation	Prove authenticity. In case of platform attestation, prove the identity of the platform.
ECALL	Enclave call. A function call that enters the enclave.
EBNF	Extended Backus–Naur Form.
EPID	Intel® Enhanced Privacy ID.
Evaluation SDK	Software development kit for evaluation purpose only.
HSM	Hardware Security Module
IAS	Intel attestation service.
LE	Launch enclave. An architectural enclave from Intel, involved in the licensing service.
ME	Manageability engine. Resides in the chipset (PCH). Amongst other features, it provides several protection related functions such as trusted time, monotonic counters and non-volatile storage. The ME is operating system independent.
Nonce	An arbitrary number used only once to sign a cryptographic communication.
OCALL	Outside call. A function call that calls an untrusted function from an enclave.
PSE	Platform service enclaves, architectural enclaves from Intel. Including PSE-pr (long-term paring) and PSE-Op (session management).

PvE	Provisioning enclave, an architectural enclave from Intel, involved in the Intel® Enhanced Privacy ID (EPID) Provision service.
QE	Quoting enclave, an architectural enclave from Intel, involved in the quoting service.
SGX	Intel® Software Guard Extensions.
SigRL	Signature revocation list
SMK	Session MAC key
SVN	Security version number. Used to version security levels of both hardware and software components of the Software Guard Extensions framework.
TCB	Trusted computing base. Portions of hardware and software that are considered safe and uncompromised. A system protection is improved if the TCB is as small as possible, making an attack harder.

Tested Environments

The Intel® Software Guard Extensions software stack – including the Intel® SGX Evaluation SDK and Platform Software (PSW) have been internally tested* by Intel and shown to work under a number of configurations. See the release notes for a list of supported environments.

Using Intel® SGX software under other environments may or may not work.

*The results have been estimated based on Intel internal analysis and are provided for informational purposes only. Any difference in system hardware or software configuration may affect actual performance.

Developing an Enclave Application

In this topic, you will see a quick guide of how to develop an enclave application. You can develop a simple enclave application after reading this topic.

Assume that you have an application with the following code:

```
#include <stdio.h>
#include <string.h>

#define MAX_BUF_LEN 100

void foo(char *buf, size_t len)
{
     const char *secret = "Hello App!";
     if (len > strlen(secret))
     {
          memcpy(buf, secret, strlen(secret) + 1);
     }
}
int main()
{
     char buffer[MAX_BUF_LEN] = "Hello World!";

     foo(buffer, MAX_BUF_LEN);
     printf("%s", buffer);

     return 0;
}
```

The program displays the string `Hello App!`

To move the function `foo` into an enclave, follow the steps below:

Step 1: Create Enclave Project

Step 2: Define Enclave Interface

Step 3: Import Enclave to Application

Step 4: Implement Application and Enclave Functions

Step 5: Compilation and Execution

Create Enclave Project

You can use Microsoft* Visual Studio* Intel® Software Guard Extensions Wizard to create an enclave project. See Step by Step Enclave Creation for the detailed steps.

Define Enclave Interface

Use an EDL file to define the enclave interface, which exposes a trusted interface `foo`. The EDL file might look like the following:

```
// sample_enclave.edl
enclave {
```

```
     trusted {
          public void foo([out, size=len] char* buf, size_t len);
     };
};
```

For detailed information about how to define the enclave interface, see the section Enclave Definition Language Syntax.

Import Enclave to Application

To call the enclave interface in the application, import the enclave to the application using Microsoft* Visual Studio* Intel® Software Guard Extensions Add-in.

1. Right click the application project and select **Intel® SGX Configuration -> Import Enclave**.
   The **Import Enclave** dialog box opens.
2. Check the **sample_enclave.edl** box and press **OK**.

Implement Application and Enclave Functions

To implement application and enclave functions, use the following code samples:

The enclave code

```
// sample_enclave.cpp
#include "sample_enclave_t.h"
#include <string.h>
void foo(char *buf, size_t len)
{
     const char *secret = "Hello Enclave!";
     if (len > strlen(secret))
     {
          memcpy(buf, secret, strlen(secret) + 1);
     }
}
```

The application code

```
#include <stdio.h>
#include <tchar.h>
#include "sgx_urts.h"
#include "sample_enclave_u.h"

#define ENCLAVE_FILE _T("sample_enclave.signed.dll")
#define MAX_BUF_LEN 100

int main()
{
     sgx_enclave_id_t   eid;
     sgx_status_t       ret   = SGX_SUCCESS;
```

```
    sgx_launch_token_t token = {0};
    int updated = 0;
    char buffer[MAX_BUF_LEN] = "Hello World!";

    // Create the Enclave with above launch token.
    ret = sgx_create_enclave(ENCLAVE_FILE, SGX_DEBUG_FLAG, &token, &up-
    dated,
                             &eid, NULL);
    if (ret != SGX_SUCCESS) {
         printf("App: error %#x, failed to create enclave.\n", ret);
         return -1;
    }


    // A bunch of Enclave calls (ECALL) will happen here.
    foo(eid, buffer, MAX_BUF_LEN);
    printf("%s", buffer);

    // Destroy the enclave when all Enclave calls finished.
    if(SGX_SUCCESS != sgx_destroy_enclave(eid))
         return -1;

    return 0;
}
```

Compilation and Execution

Now you can compile the application and enclave projects. After the compilation, set the working directory to the output directory and run the program. You should get the string `Hello Enclave!`

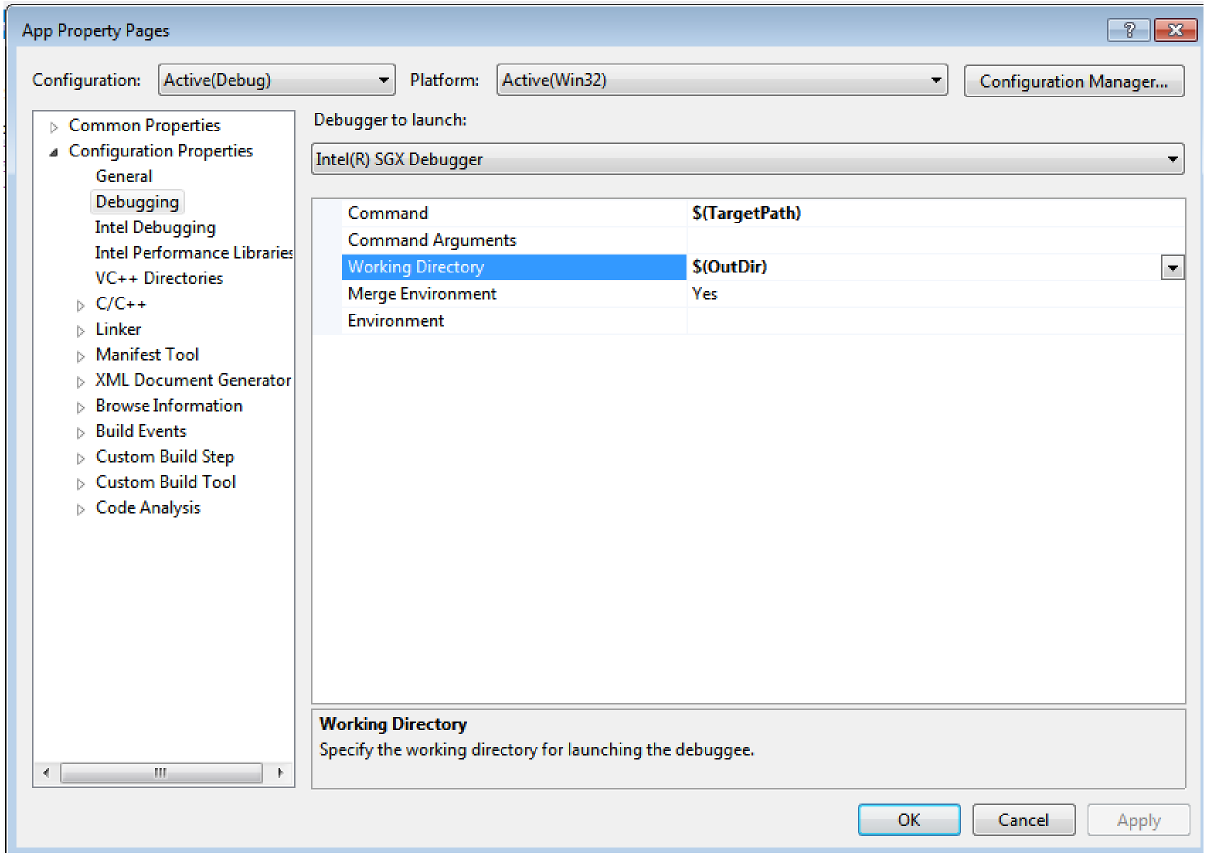

Figure 1 Setting Working Directory in Microsoft* Visual Studio*

Setting up an Intel® Software Guard Extension Project

This topic introduces how to use the following features of Intel® Software Guard Extensions Evaluation SDK:

- Using Microsoft* Visual Studio* Intel® Software Guard Extensions Wizard
- Using Microsoft* Visual Studio* Intel® Software Guard Extensions Add-in
- Enclave Project Files
- Project Settings

Using Microsoft* Visual Studio* Intel® Software Guard Extensions Wizard

Intel® Software Guard Extensions Evaluation SDK installs a Microsoft* Visual Studio* software wizard to aid developers in rapid development of Intel® Software Guard Extensions. This wizard can be used to create an enclave project, which then has the proper settings to take advantage of the various components that are shipped with the Intel® SGX Evaluation SDK.

Step by Step Enclave Creation

1. On the menu bar of Microsoft* Visual Studio*, choose **File-->New-->Project**.
   The **New Project** dialog box opens.
2. Select **Templates-->Visual C++-->Intel® SGX Enclave Project**. Enter name, location, and solution name in the appropriate fields like any other Microsoft* Visual Studio* project.

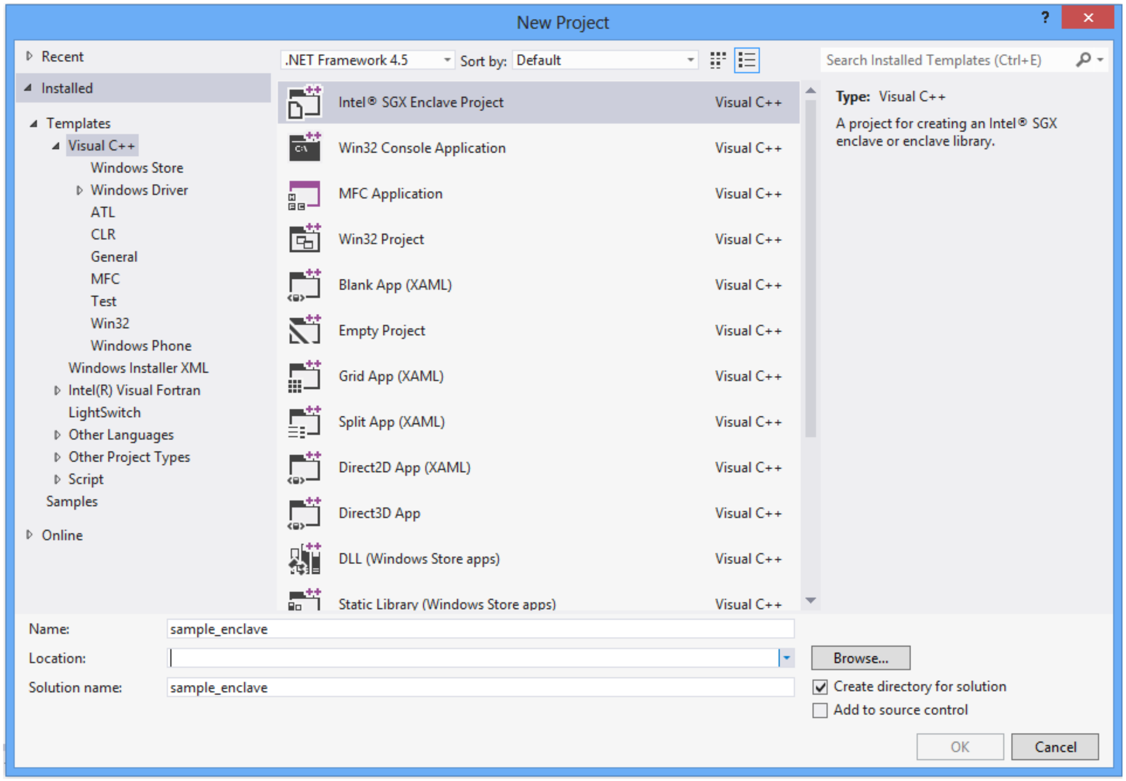

Figure 2 Intel® SGX Wizard: New Project Creation

3. Click **OK** and the welcome dialog appears.

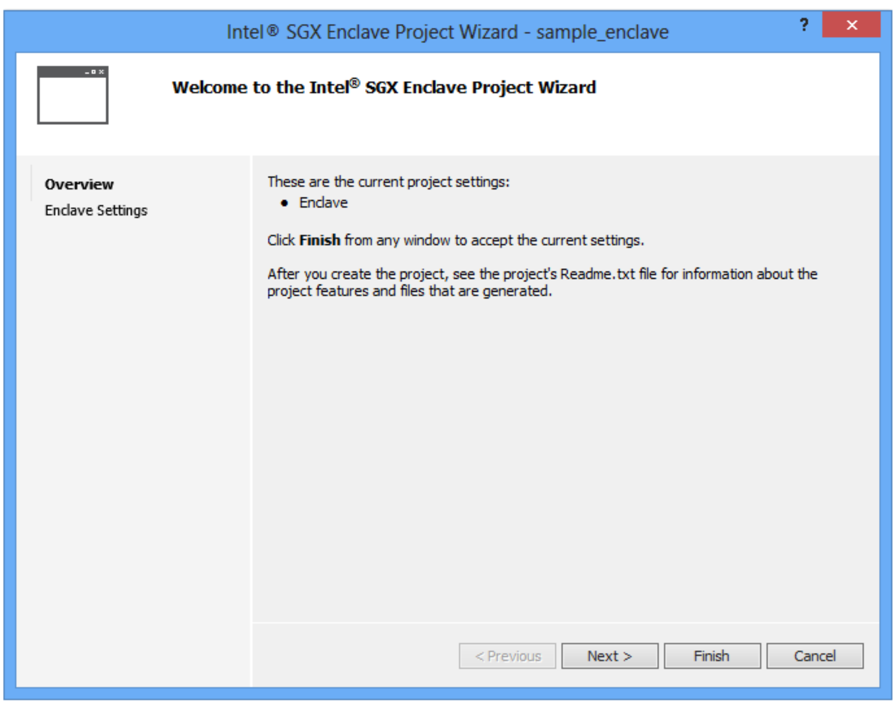

Figure 3 Intel® SGX Wizard: Welcome Dialog

4. Click **Next** to go to the Enclave Settings page.

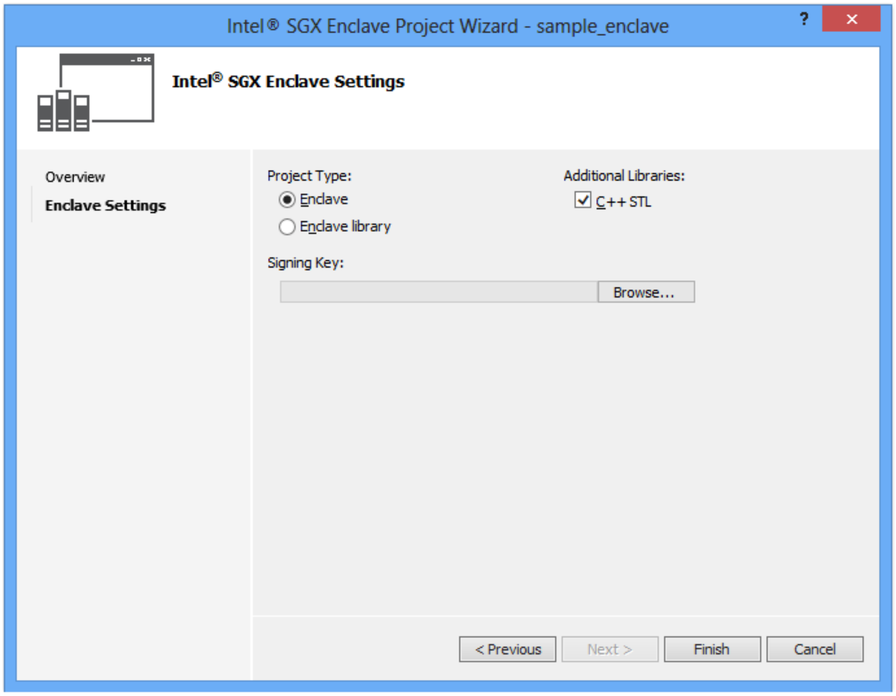

Figure 4 Intel® SGX Wizard: Enclave Settings

5. Configure the enclave with proper settings
   - Project Type:
     - Enclave – Create an enclave project.
     - Enclave library – Create a static library for an enclave project.
   - Additional Libraries:
     - C++ STL – Link C++ STL with the enclave project.
   - Signing Key:
     - Import an existing signing key to the enclave project. A random key will be generated if no file is selected. The Enclave signer will sign the enclave with the key file (see File Formats).

   When the enclave project is created, the wizard ensures that the enclave project has proper settings.

NOTE:

The Wizard creates an enclave project with several files. See Enclave Project Files for a detailed file list.

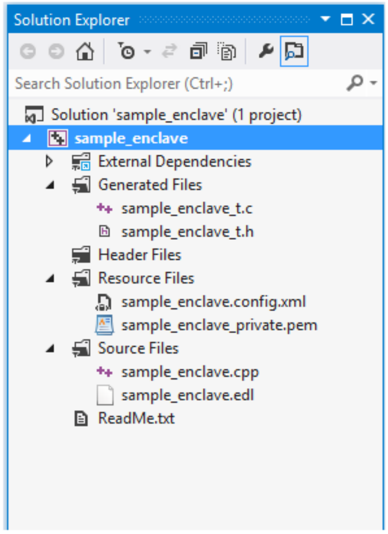

Figure 5 Intel® SGX Wizard: Solution Explorer

Add 64-bit Build Support (Optional)

1. Right click on solution name and select **Configuration Manager**.
2. From the **Active solution platform** combo box, select **<New>**.
3. Select **x64** and press **OK**.

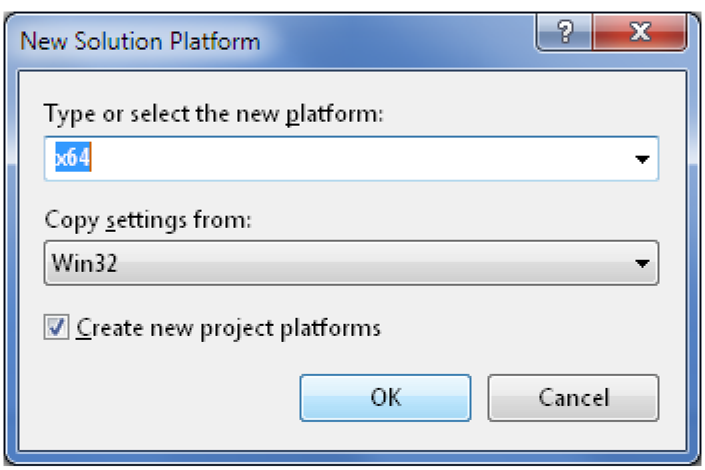

Figure 6 Solution's New Configuration

4. Save the solution's new configuration.

Using Microsoft* Visual Studio* Intel® Software Guard Extensions Add-in

The Microsoft* Visual Studio* add-in is provided to the Intel® Software Guard Extensions developer for configuring an enclave or importing an enclave to untrusted components conveniently and efficiently. This add-in has three main features:

- Enclave Settings helps to maintain the enclave configuration settings
- Enclave Signing helps to perform enclave two-step signing for release mode.
- Import Enclave helps to select the enclaves to be imported to the untrusted components. Then the untrusted components can make use with the enclave.

Enclave Settings

Enclave settings helps you to create and maintain the enclave configuration file. The enclave configuration file is part of the enclave project and describes the information of the enclave metadata. See Enclave Configuration File for details.

Enclave Settings gives the user the option to update the following enclave settings:

- ProdID
- ISVSVN
- StackMaxSize
- HeapMaxSize
- TCSNum
- TCSPolicy
- DisableDebug
- MiscSelect
- MiscMask

To configure enclave settings:

Open the solution that contains the enclave project. Right click the enclave project. Select **Intel® SGX Configuration -> Enclave Settings**. A dialog will be shown which allows the modification of the enclave settings. Here is a sample of the dialog.

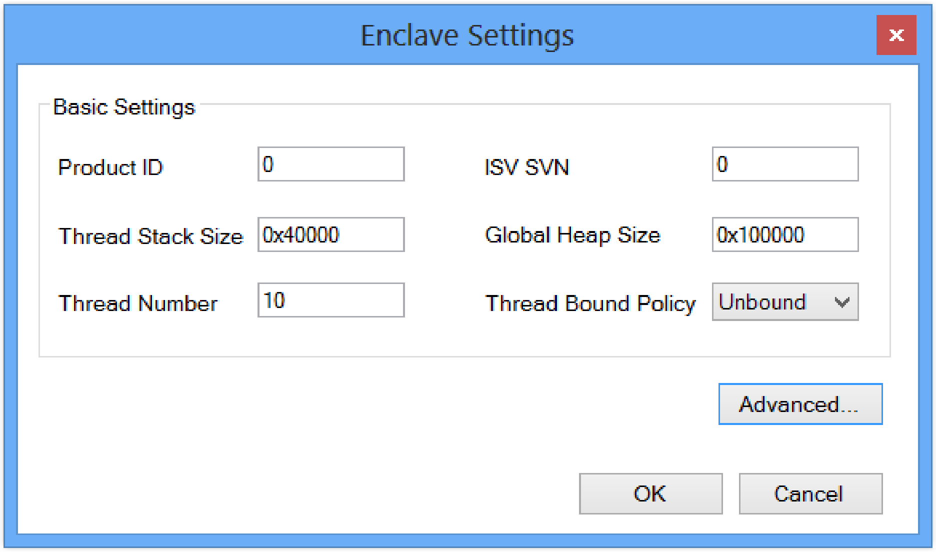

Figure 7 Intel® SGX Configuration: Enclave Settings

The **Basic Settings** box provides users the interface to modify the basic enclave settings. The following list gives an explanation of each configuration element.

Name	Description	Tag in the Enclave Configuration File
Product ID	ISV assigned Product ID	`<ProdID>`
ISV SVN	ISV assigned SVN	`<ISVSVN>`
Thread Stack Size	The stack size per trusted thread (in bytes)	`<StackMaxSize>`
Global Heap Size	The heap size for the enclave (in bytes)	`<HeapMaxSize>`
Thread Number	The number of trusted threads	`<TCSNum>`
Thread Bound Policy	TCS management policy	`<TCSPolicy>`

Table 1 Settings in the Enclave Configuration File

The **Advanced Settings** dialog shows the interface to modify the advanced features. Given that users have enough knowledge of these advanced features, click the button **Advanced...**, then the following window appears:

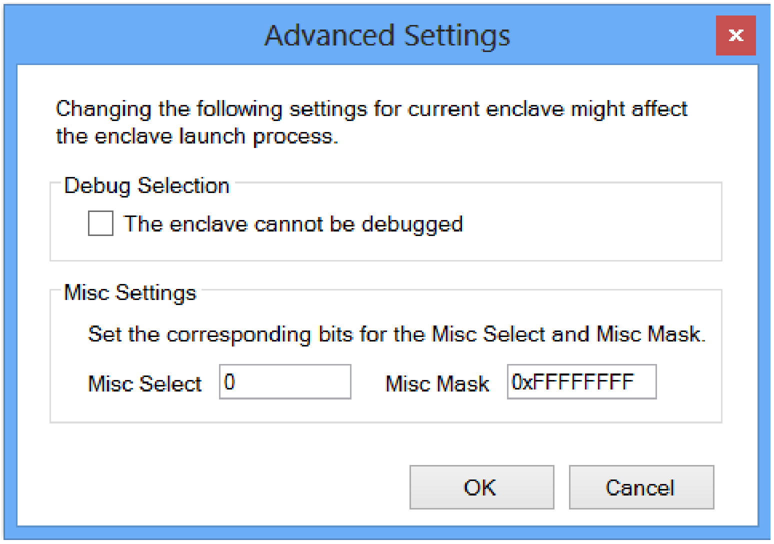

Figure 8 Intel® SGX Configuration: Advanced Enclave Settings

Check the Debug selection if you want to ensure the enclave cannot be launched in debug mode. The setting corresponds to the element `<DisableDebug>` of the Enclave Configuration File. The code/data memory inside an enclave launched in debug mode is accessible by the debugger or other software outside of the enclave. Thus, it does not have the same memory access protection as an enclave launched in non-debug mode. An enclave can only be debugged if it is launched in debug mode. If the selection is checked, the enclave built with this configuration cannot be debugged.

You can set the bits value for the Misc Select and Misc Mask in the **Advanced Settings** dialog. These settings respectively correspond to the element `<MiscSelect>` and `<MiscMask>` of the Enclave Configuration File. The `<MiscSelect>` and `<MiscMask>` are for functionality extension in the future. Currently only `0` for can be set for Misc Select by default. The recommendation is keeping the default settings.

Enclave Signing

With the enclave launch control, the enclave signing key for release mode must be stored in HSM. All the release mode enclaves should use two-step signing mechanism. Enclave Signing Examples describes a command line example for this two-step signing process. **Enclave Signing** provides a GUI to help developers to perform the two-step signing process more easily and more conveniently.

Step 1: Generate Enclave Signing Material

Open the solution that contains the enclave project. Right click the enclave project. Select **Intel® SGX Configuration -> Enclave Signing**. The **Enclave Signing** dialog appears. The following graphic shows a GUI sample for generating the enclave signing material.

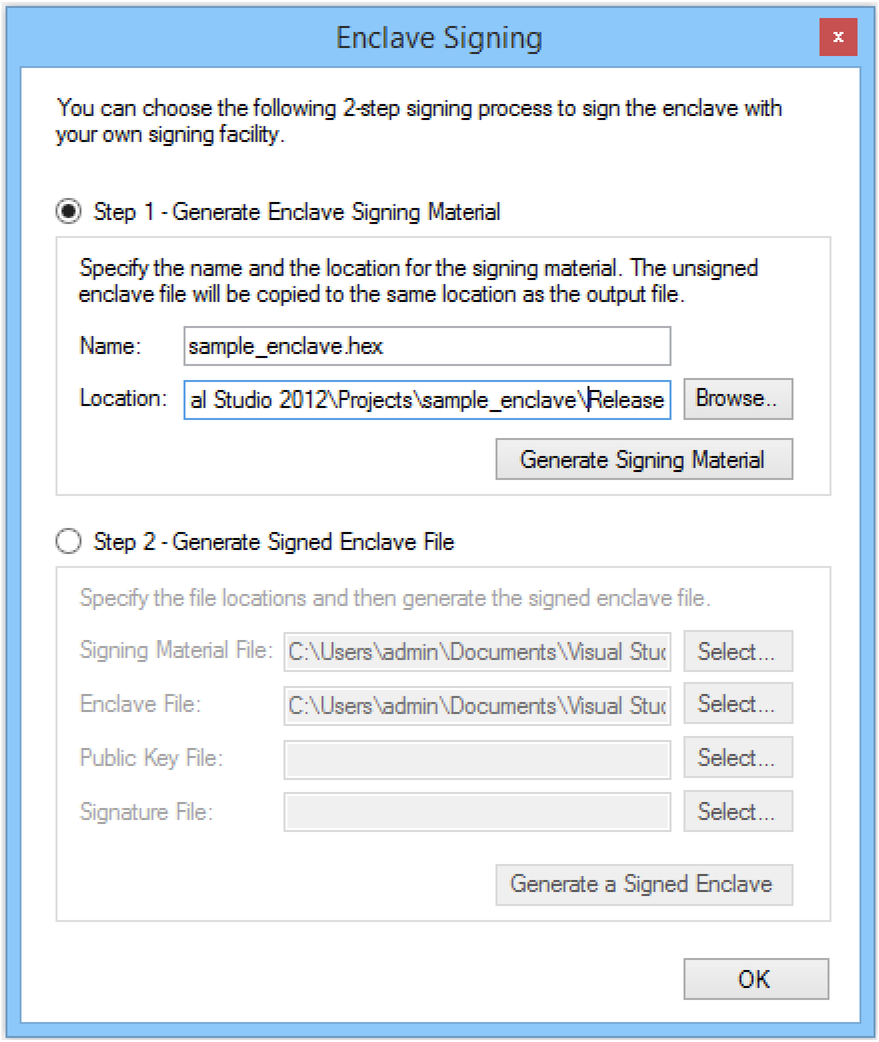

Figure 9 Intel® SGX Configuration: Generate Enclave Signing Material

The default name and location for the output enclave signing material are specified. You can change the name and location. Click the button **Generate Signing Material** to generate the enclave signing material.

After finishing Step 1, you need to use your own signing facility which can access your private signing key to sign the output enclave signing material, and take the resulting signature file back for Step 2.

NOTE

By default, a **Post Build Event** for generating enclave signing material is added during enclave creation with Microsoft* Visual Studio* in Intel® Software Guard Extensions Wizard. Thus, for the release mode, the enclave signing material is generated automatically after you compile the enclave project.

Step 2: Generate a Signed Enclave File

If you have finished generating the enclave signing material and have prepared the resulting signature file, you can generate the signed enclave file. To generate the final signed enclave file, select the radio button next to **Step 2 - Generate Signed Enclave File**.

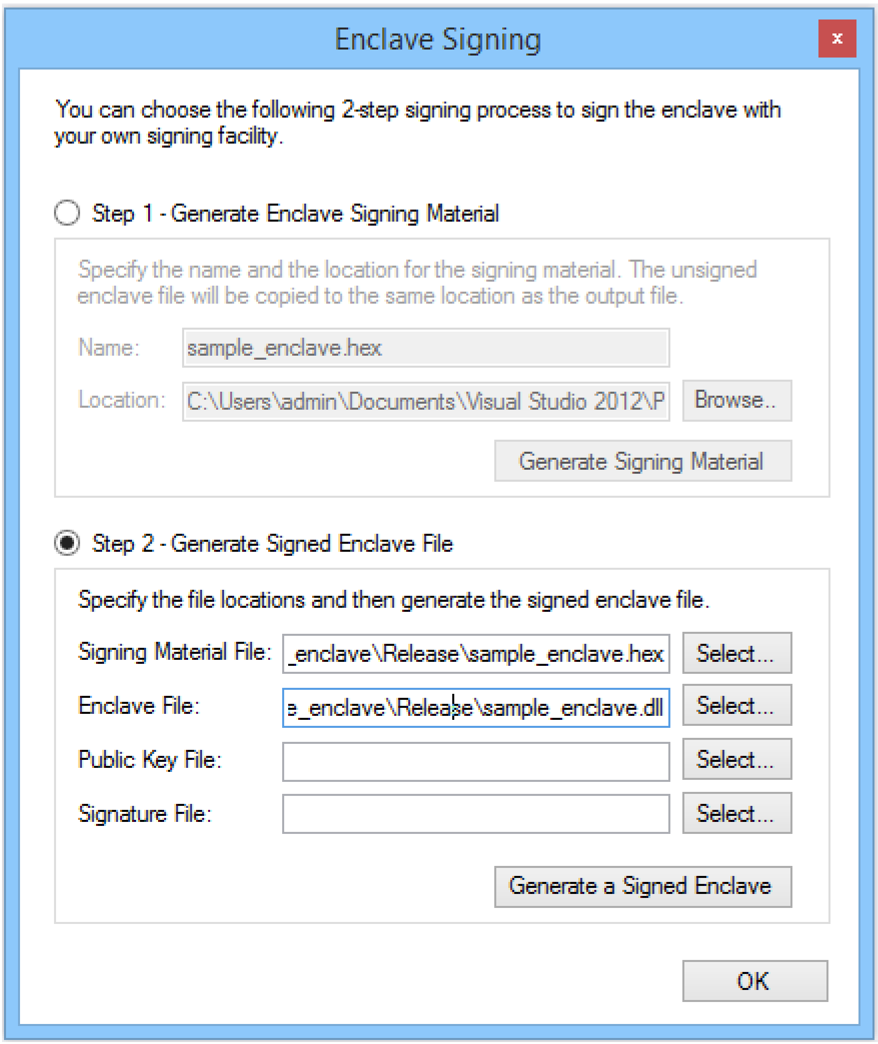

Figure 10 Intel® SGX Configuration: Generate a Signed Enclave File

The default location for the signing material and the enclave file are specified. Check if the specified pathes are correct for the signing material and the enclave file. Click the button **Select...** next to **Public Key File** to specify the corresponding public key. Click the button **Select...** next to **Signature File** to specify the resulting signature file.

After specifying all the correct files, click the button **Generate Signed Enclave**, then the final signed enclave file is generated under the same folder of the specified enclave file.

Import Enclave

Import Enclave helps to select the enclaves to be imported to the untrusted components. Then the untrusted components can make use with the enclaves.

Import Enclave provides the following functions:

- Allows selecting an enclave from the list of enclaves created with the Intel® SGX Wizard in the same solution.
- Supports browsing/searching for 3rd party provided enclaves which are defined by EDL files.
- Provides the option to remove any enclave selected to be hosted by an application.
- Adds/removes the enclave's _u.h and _u.c files to/from the untrusted component project, for each enclave that is added to/removed from the application.
- Sets up the project settings for the untrusted component.

To import enclaves:

Open the solution that contains both the enclave project and the untrusted application project that will host the enclave. Right click the untrusted application project. Select **Intel® SGX Configuration -> Import Enclave**. The following sample dialog will be shown. In this example, the enclave project name is **sample_enclave** and the hosting project is a Win32 console application.

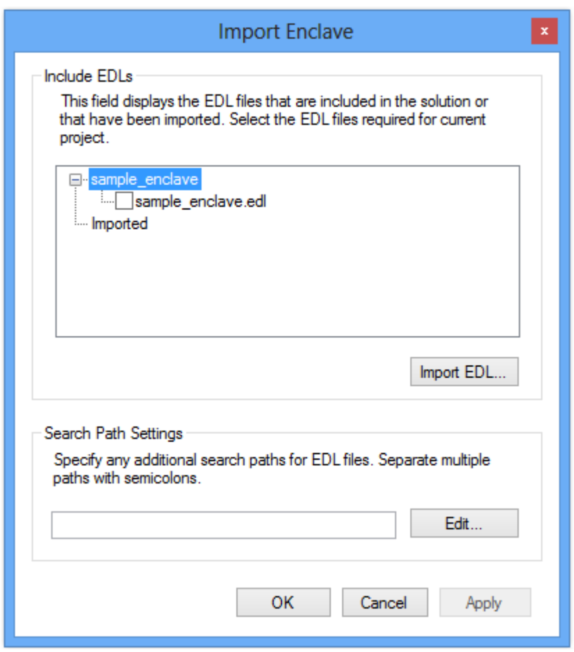

Figure 11 Intel® SGX Configuration: Import Enclave

The **Include EDLs** section in the **Import Enclave** dialogue contains all the enclaves in this solution and any enclaves imported from outside the solution. Each enclave is represented by an Enclave Definition Language (EDL) file. EDL is a minimal IDL used to describe the enclave interface. See Enclave Definition Language Syntax for a detailed description. Select the EDL files corresponding to the enclaves to be imported into the application.

To import an enclave that is not in the solution, you can click **Import EDL...** to select a new EDL file. The imported enclaves are listed in the **Imported** field. You need to select any of the imported EDL files representing the enclaves you wish to import into the application.

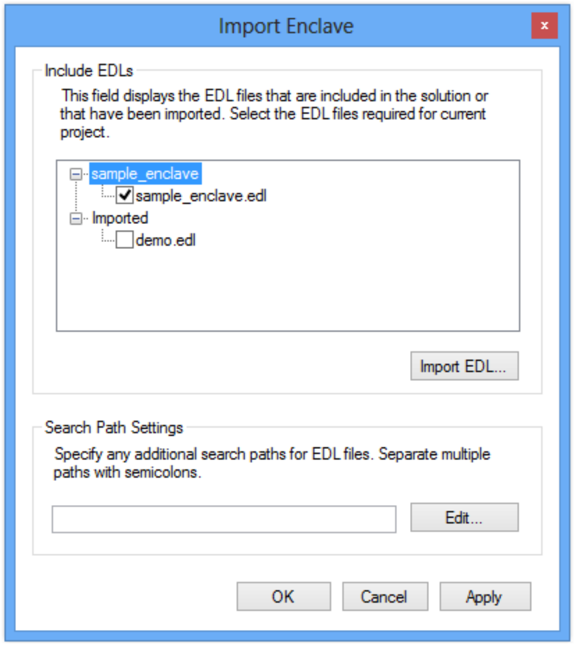

Figure 12 Intel® SGX Configuration: Import EDL File

If the selected EDL files require additional search paths for any embedded EDL files, specify the search paths in the **Search Path Settings**.

To put the actions into effect, click **OK**. Then two files will be added to the untrusted application project: `sample_enclave_u.c` and `sample_enclave_u.h`. They contain the declarations and definitions of the untrusted wrapper functions used to call enclave functions. In addition, the properties of the untrusted application project are modified to recreate the files when the project is rebuilt. The files are not expected to be modified by the user. To cancel the actions, click **Cancel**.

To remove an imported enclave from the untrusted application, unselect the corresponding EDL file and click **OK**. Then the corresponding settings in the untrusted application are removed.

Enclave Project Files

The Intel® Software Guard Extensions wizard is used to create enclave projects. It creates several files with names derived from the project name.

Assuming the enclave project name is *sample_enclave*, here is the list of files generated by the wizard.

Source files:

- `sample_enclave.cpp` – main source file, to be filled with user functions and variables. The user can add additional source files.
- `sample_enclave_t.c` – trusted auto-generated wrapper functions. Do not modify this file as every build recreates it.
- `sample_enclave.edl` – enclave definition language (EDL) file. Declares which functions are exported (trusted) and imported (untrusted) by the enclave. EDL syntax is explained in a separate chapter.

Header files:

- `sample_enclave_t.h` – trusted auto-generated header for wrapper functions. Do not modify this file as every build recreates it.

Resource files:

- `sample_enclave.config.xml` – specifies the enclave configuration. Details are explained in a separate section.
- `sample_enclave.private.pem` – RSA private key used to sign the enclave.

NOTE:

The private key must be kept secret and safe. If it is exposed, the key could be used by malware writers to create a valid signed enclave.If you do not want to expose the private key in the enclave project, you can use sgx_sign to sign the enclave in a separate environment. See The Enclave Signing Tool for a detailed description.

Project Settings

This section introduces the following project settings:

- Recommended Project Settings for an Enclave Project
- Recommended Project Settings for an Untrusted Application

To configure the project settings in Microsoft* Visual Studio*, right click the project name in **Solution Explorer** and select **Properties** from the context menu.

Recommended Project Settings for an Enclave Project

For an enclave, default project settings are recommended, with the following exceptions:

C/C++->General->Additional Include Directories:

```
$(SGXSDKInstallPath) include;$(SGXSDKInstallPath) include\t-libc;$(SGXSDKInstallPath)include\stlport;
```

Linker->General->Additional Library Directories:

```
$(SGXSDKInstallPath)bin\$(Platform)\$(Configuration)
```

Linker->Input->Additional Dependencies:

HW Configuration: `sgx_ trts.lib; sgx_ tservice.lib; sgx_ tstdc.lib; sgx_ tstdcxx.lib; sgx_tcrypto_opt.lib`

Simulation Configuration: `sgx_ trts.lib_ sim; sgx_ tservice_ sim.lib; sgx_ tstdc.lib; sgx_tstdcxx.lib; sgx_tcrypto.lib`

Linker->Input->Ignore All Default Libraries:`Yes (/NODEFAULTLIB)`

Linker->Advanced->No Entry Point:`Yes`

To sign the enclave during the build process, set a custom build step in the project settings:

Build Events->Post-Build Event->Command Line:

```
"$(SGXSDKInstallPath)bin\win32\release\sgx_ sign.exe" sign - key "$(ProjectDir)sample_enclave_private.pem" -enclave "$(OutDir)sample_enclave.dll" -out "$(OutDir)sample_enclave.signed.dll" -config "$(ProjectDir)sample_enclave.config.xml"
```

Build Events->Post-Build Event->Use In Build:`Yes`

NOTE:

The signing command line is a sample command. Change the command line based on the actual enclave name.

NOTE:

A few compiler options are not supported when enclave code is compiled. See sections Unsupported MSVC* Options for Enclaves and Unsupported Intel® Compiler Options for Enclaves for a detailed list.

Recommended Project Settings for an Untrusted Application

Use the default project settings for an untrusted application, with the following additional settings:

C/C++->General->Additional Include Directories: `$(SGXSDKInstallPath)include;`

Linker->General->Additional Library Directories: `$(SGXSDKInstallPath)bin\$(Platform)\$(Configuration)`

Linker->Input->Additional Dependencies:

HW Configuration: `sgx_uae_service.lib; sgx_urts.lib`

Simulation Configuration: `sgx_uae_service_sim.lib; sgx_urts_sim.lib`

Supported Application Types

The Intel® Software Guard Extensions Evaluation SDK supports a number of application types and user accounts on the Windows* OS. Users of regular, guest and administrator accounts may run an enclave application in the form of a DLL to load and interface with an Intel® SGX enclave. User-level driver and system services that execute in the security context of a user account also have access to the functionality provided by the Intel® SGX software stack.

Using Intel® Software Guard Extensions Evaluation SDK Tools

This topic introduces how to use the following tools that the Intel® Software Guard Extensions Evaluation SDK provides:

- The Edger8r Tool
  Generates interfaces between the untrusted components and enclaves.
- The Enclave Signing Tool
  Generates the enclave metadata, which includes the enclave signature, and adds such metadata to the enclave image.
- Enclave Debugger
  Helps to debug an enclave.
- Enclave Memory Measurement Tool
  Helps to measure the usage of protected memory by the enclave at runtime.
- CPUSVN Configuration Tool
  Helps to simulate the CPUSVN upgrade/downgrade scenario without modifying the hardware.

The Edger8r Tool

The Edger8r tool ships as part of the Intel® Software Guard Extensions Evaluation SDK. It generates edge routines by reading a user-provided EDL file. These edge routines provide the interface between the untrusted application and the enclave. Normally, the tool will run automatically as part of the enclave build process. However, an advanced enclave writer may invoke the Edger8r manually.

When given an EDL file, for example, `demo.edl`, the Edger8r will by default generate four files:

- `demo_t.h` – It contains prototype declarations for trusted proxies and bridges.
- `demo_t.c` – It contains function definitions for trusted proxies and bridges.
- `demo_u.h` – It contains prototype declarations for untrusted proxies and bridges.
- `demo_u.c` – It contains function definitions for untrusted proxies and bridges.

Here is the usage description for the Edger8r tool:

Syntax:

```
sgx_edger8r [options] <.edl file> [another .edl file …]
```

Arguments:

[Options]	Descriptions
`--use-prefix`	Prefix the untrusted proxy with the enclave name.
`--header-only`	Generate header files only.
`--search-path <path>`	Specify the search path of EDL files.
`--untrusted`	Generate untrusted proxy and bridge routines only.
`--trusted`	Generate trusted proxy and bridge routines only.

`--untrusted-dir <dir>`	Specify the directory for saving the untrusted code.
`--trusted-dir <dir>`	Specify the directory for saving the trusted code.
`--help`	Print this help message.

If neither `--untrusted` nor `--trusted` is specified, the Edger8r will generate both.

Here, the `path` parameter has the same format as the PATH environment variable, and the enclave name is the base file name of the EDL file (`demo` in this case).

CAUTION:

The ISV must run the Edger8r tool in a protected malware-free environment to ensure the integrity of the tool so that the generated code is not compromised. The ISV is ultimately responsible for the code contained in the enclave and should review the code that the Edger8r tool generates.

The Enclave Signing Tool

The Intel® Software Guard Extensions Evaluation SDK provides a tool named *sgx_sign* for you to sign enclaves. In general, signing an enclave is a process that involves producing a signature structure that contains enclave properties such as the enclave measurement. Once an enclave is signed in such structure, the modifications to the enclave file (such as code, data, signature, and so on.) can be detected. The signing tool also evaluates the enclave image for potential errors and warns users about potential security hazards. *sgx_sign* is typically set up by one of the configuration tools included in the Intel® SGX SDK and runs automatically at the end of the build process. During the loading process, the signature is checked to confirm that the enclave has not been tampered with and has been loaded correctly.

Command-Line Syntax

To run *sgx_sign*, use the following command syntax:

```
sgx_sign <command> [args]
```

All valid commands are listed in the table below. See Enclave Signing Examples for more information.

Table 2 Signing Tool Commands

Command	Description	Arguments
`sign`	Sign the enclave using the private key in one step.	Required: -enclave, -key, -out Optional: -config
`gendata`	The first step of the 2-step signing process. Generate the enclave signing material to be signed by an external tool. This step dumps the signing material, which consists of the header and body sections of the enclave sig-	Required: -enclave, -out Optional: -config

	nature structure (see the Table Enclave Signature Structure in this topic), into a file (256 bytes in total).	
`catsig`	The second step of the 2-step signing process. Generate the signed enclave with the input signature and public key. The input signature is generated by an external tool based on the data generated by the `gendata` command. At this step, the signature and buffer sections are generated. The signature and buffer sections together with the header and body sections complete the enclave signature structure (see the Table Enclave Signature Structure in this topic).	Required: -enclave, -key, -out, -sig, -unisgned Optional: -config

All the valid command options are listed below:

Table 3 Signing Tool Arguments

<table>
<tr><th>Arguments</th><th>Descriptions</th></tr>
<tr><td><code>-enclave <file></code></td><td>Specify the enclave file to be signed.
It is a required argument for the three commands.</td></tr>
<tr><td><code>-config <file></code></td><td>Specify the enclave configuration file
It is an optional argument for the three commands.</td></tr>
<tr><td><code>-out <file></code></td><td>Specify the output file.
It is required for the three commands.
<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td><code>sign</code></td><td>The signed enclave file.</td></tr>
<tr><td><code>gendata</code></td><td>The file with the enclave signing material.</td></tr>
<tr><td><code>catsig</code></td><td>The signed enclave file.</td></tr>
</table></td></tr>
<tr><td><code>-key <file></code></td><td>Specify the signing key file. See File Formats for detailed description.
<table>
<tr><td>Command</td><td>Description</td></tr>
<tr><td><code>sign</code></td><td>Private key.</td></tr>
<tr><td><code>gendata</code></td><td>Not applicable.</td></tr>
<tr><td><code>catsig</code></td><td>Public key.</td></tr>
</table></td></tr>
<tr><td><code>-sig <file></code></td><td>Specify the file containing the signature corresponding to the enclave</td></tr>
</table>

	signing material. Only valid for `catsig` command.
`-unsigned <file>`	Specify the file containing the enclave signing material generated by `gendata`. Only valid for `catsig` command.

The arguments include options and filenames and can be specified in any order. Options are processed first, then filenames. Use one or more spaces or tabs to separate arguments. Each option consists of an option identifier, a dash (-), followed by the name of the option. The `<file>` parameter specifies the absolute or relative path of a file.

Users can start *sgx_sign* from a system command prompt or integrate the command line into a **Post Build Event** under the enclave project properties in Microsoft* Visual Studio* IDE. To follow the different command character set rules in different platforms, sgx_sign Command-Line is case-insensitive in Windows* OS.

sgx_sign generates the output file and returns 0 for success. Otherwise, it generates an error message and returns -1.

Table 4 Enclave Signature Structure

Section	Name
Header	HEADERTYPE
	HEADERLEN
	HEADERVERSION
	TYPE
	MODVENDOR
	DATE
	SIZE
	KEYSIZE
	MODULUSSIZE
	ENPONENTSIZE
	SWDEFINED
	RESERVED
Signature	MODULUS
	EXPONENT
	SIGNATURE

Section	Name
Body	MISCSELECT
	MISCMASK
	RESERVED
	ATTRIBUTES
	ATTRIBUTEMASK
	ENCLAVEHASH
	RESERVED
	ISVPRODID
	ISVSVN
Buffer	RESERVED
	Q1
	Q2

Enclave Signing Key Management

An enclave project supports different signing methods needed by ISVs during the enclave development life cycle.

- Single-step method using the ISV's test private key:
  The signing tool supports a single-step signing process, which requires the access to the signing key pair on the local build system. However, there is a requirement that any white-listed enclave signing key must be managed in a hardware security module. Thus, the ISV's test private key stored in the build platform will not be white-listed and enclaves signed with this key can only be launched in *debug* or *prerelease* mode. In this scenario, the ISV manages the signing key pair, which could be generated by the Microsoft* Visual Studio Wizard when the enclave project is created or by the ISV using his own means. Single-step method is the default signing method for non-production enclave applications, which are created with the Intel SGX project *debug* and *prerelease* profiles.
- 2-step method using an external signing tool:
  1. First step: At the end of the enclave build process, the signing tool generates the enclave signing material. The ISV may also generate the enclave signing material file by an option available in the Microsoft* Visual Studio Add-in.
     The ISV takes the enclave signing material file to an external signing platform/facility where the private key is stored, signs the signing material file, and takes the resulting signature file back to the build platform.
  2. Second step: The ISV selects the *Second Step Signing* option from the Microsoft* Visual Studio Add-in to add the hash of the public key and signature to the enclave's metadata section.

  The 2-step signing process protects the signing key in a separate facility. Thus it is the default signing method for the Intel SGX project *release* profile. This means it is the only method for signing production enclave applications.

File Formats

There are several files with various formats followed by the different options. The file format details are listed below.

Table 5 Signing Tool File Formats

File	Format	Description
Enclave file	DLL	It is a standard DLL.
Signed enclave file	DLL	`sgx_sign` generates the signed enclave file , which includes the signature, to the enclave file.
Configuration file	XML	See Enclave Configuration File.
Key file	PEM	Key file should follow the PEM format which contains an unencrypted RSA 3072-bit key. The public exponent must be 3.
Enclave hex file	RAW	It is a dump file of the enclave signing material data to be signed with the private RSA key.
Signature file	RAW	It is a dump file of the signature generated at the ISV's signing facility. The signature should follow the RSA-PKCS1.5 padding scheme. The signature should be generated using the v1.5 version of the RSA scheme with an SHA-256 message digest.

Signing Key Files

The enclave signing tool only accepts key files in the PEM format and unencrypted. When an enclave project is created for the first time, you have to choose between using an already existing signing key or automatically generating one key for you. When you choose to import a pre-existing key, ensure that such key is in PEM format and unencrypted. If that is not the case, convert the signing key to the format accepted by the Signing Tool first. For instance, the following command converts an encrypted private key in PKCS#8/DER format to unencrypted PEM format:

```
openssl pkcs8 -inform DER -in private_pkcs8.der -outform PEM -out private_
pkcs1.pem
```

Depending on the platform OS, the openssl* utility might be installed already or it may be shipped with the Intel® SGX SDK.

Enclave Signing Examples

The following are typical examples for signing an enclave using the one-step or the two-step method. When the private signing key is available at the build platform, you may follow the one-step signing process to sign your enclave. However, when the private key is only accessible in an isolated signing facility, you must follow the two-step signing process described below.

- One-step signing process:
  Signing an enclave using a private key available on the build system:
  ```
  sgx_sign sign -enclave enclave.dll -config config.xml -out enclave_
  signed.dll -key private.pem
  ```

- Two-step signing process:
  Signing an enclave using a private key stored in an HSM, for instance:
  1. Generate the enclave signing material.
     `sgx_sign gendata -enclave enclave.dll -config config.xml -out enclave_hash.hex`
  2. At the signing facility, sign the file containing the enclave signing material (`enclave_hash.hex`) and take the resulting signature file (`signature.hex`) back to the build platform.
  3. Sign the enclave using the signature file and public key.
     `sgx_sign catsig -enclave enclave.dll -config config.xml -out enclave_signed.dll -key public.pem -sig signature.hex -unsigned enclave_hash.hex`

The configuration file `config.xml` is optional. If you do not provide a configuration file, the signing tool uses the default configuration values.

A single enclave signing tool is provided, which allows signing 32-bit and 64-bit enclaves. In addition, on Windows* OS `sgx_sign` supports signing enclaves in both PE and ELF formats.

OpenSSL* Examples

The following command lines are typical examples using OpenSSL*.

1. Generate a 3072-bit RSA private key. Use `3` as the public exponent value.

```
openssl genrsa -out private_key.pem -3 3072
```

2. Produce the public part of a private RSA key.

```
openssl rsa -in private_key.pem -pubout -out public_key.pem
```

3. Sign the file containing the enclave signing material.

```
openssl dgst -sha256 -out signature.hex -sign private_key.pem -keyform PEM
enclave_hash.hex
```

Enclave Debugger

Only a debug mode enclave can be debugged with the Intel® SGX debugger. First, the enclave must be built as debuggable. See Enclave Settings to unselect **Ensure the enclave cannot be launched in debug mode** in the enclave Advanced Configuration. Second, in the application, the enclave must be loaded in debug mode. To load an enclave in debug mode, the debugger flag (the second parameter of sgx_create_enclave) must be TRUE.

To utilize the Intel® SGX debugger to debug an enclave, you must change the Microsoft* Visual Studio* project properties for the enclave. Right click the enclave project and select **Project Properties**. Go to **Properties->Configuration Properties->Debugging** and select **Intel(R) SGX Debugger** as shown below.

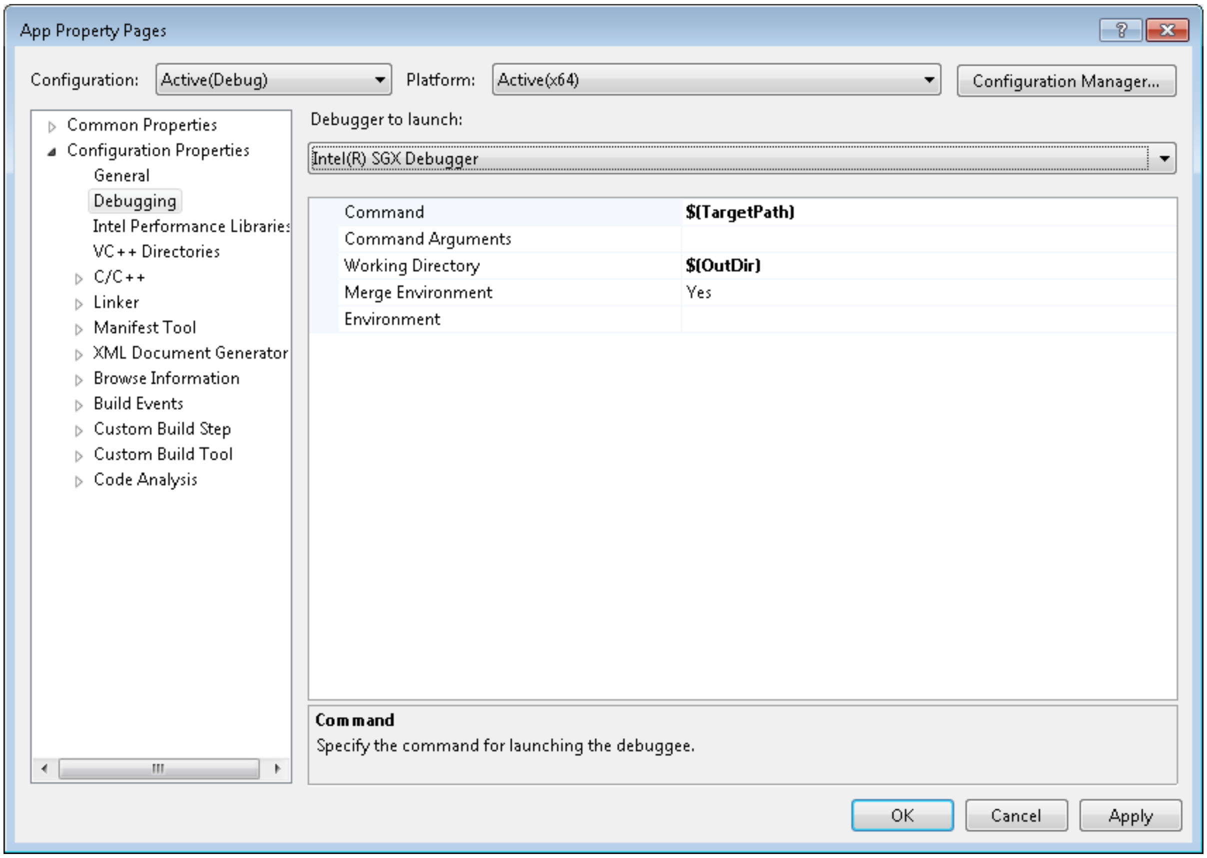

Figure 13 Intel® SGX Debugger Enabling

Starting and Debugging an Enclavized Application from within Microsoft* Visual Studio

Once the Intel® SGX Debugger has been selected for the enclave project, setting breakpoints and/or stepping into an enclave works exactly as normal application debugging does in Microsoft* Visual Studio*.

Attaching to and Debugging an Enclave inside a Running Process

Use the following steps to attach to and debug an enclave inside a running process:

1. In Microsoft* Visual Studio*, select **DEBUG-> Attach to process->Select->Intel® SGX code**.
2. Highlight the process that you would like to attach to and debug; then click the **Select** button.
3. In the pop-up dialog, **Select Code Type**, select **Debug these code types**; then check **Intel® SGX**.
4. Click **OK** in the **Select Code Type** dialog, and click **Attach** in the **Attach to Process** dialog.

When the Intel® SGX Debugger is used as remote debugger, the host machine needs both Intel® SGX SDK and Intel® SGX Debugger installed, and the target machine needs the Intel® SGX Debugger and Microsoft* Visual Studio* remote debugger server (see Remote Debugging Setup at http://msdn.microsoft.com/en-us/library/vstudio/y7f5zaaa.aspx).

First, launch the Remote Debugging Monitor (`msvsmon.exe`) on the target machine. (See details about Start the Remote Debugging Monitor at http://msdn.microsoft.com/en- us/library/vstudio/xf8k2h6a.aspx)

On the host machine, select **DEBUG-> Attach to process->Select->Intel® SGX code** in Microsoft* Visual Studio* and set the qualifier as the target machine name or IP address.

The Intel® SGX Debugger can be used to debug both an enclave project and an untrusted application, but cannot be used to debug the uRTS and tRTS, which are part of Intel® Software Guard Extensions Evaluation SDK. When a breakpoint occurs inside the uRTS or tRTS, the debugger is not able to display any symbols and the button **step out** does not work. To fix this issue, manually add one more break point outside the uRTS and tRTS.

The Intel® SGX Debugger only supports native C/C++ code. It is not able to debug managed code or native/managed code mix mode. If the enclave is used in mix mode, the only way to debug it is by using the debugger attach feature.

Enclave Memory Measurement Tool

An enclave is an isolated environment. Popular performance analysis tools (such as AQtime*, perf tools, VTune*, Valgrind*, and so on), are not supported inside an enclave. The Intel® Software Guard Extensions Evaluation SDK provides a tool called `sgx_emmt` to measure the real usage of protected memory by the enclave at runtime.

Currently the enclave memory measurement tool provides the following two functions:

1. Get the stack peak usage value for the enclave.
2. Get the heap peak usage value for the enclave.

When you get the accurate stack and heap usage information for your enclaves, you can rework the enclave configuration file based on this information to make full use of the protected memory. See Enclave Configuration File for details.

The tool is a separate application under Windows* OS. To measure the protected memory consumption by one enclave, leverage this tool to launch a test application which in turns loads the enclave. Use the following syntax for `sgx_emmt`:

```
sgx_emmt [--enclave=<enclave list>] application_name <application args>
```

Arguments:

--enclave:

This is an optional argument. It follows the measurement targets which are specified by `<enclave list>`. If users do not provide this parameter, the tool will collect the protected memory usage information for each measurable enclave. If more than one enclave needs to be measured, all the enclave names should be listed in `<enclave list>` separated by comma (,) without any blank space.

application:

It is the required argument which indicates the test application name. The application arguments are provided in `<application args>` if there are any.

Examples:

Assume a test application name is `myApp` with two input parameters. The test application manages three enclaves named `myEnclave1`, `myEnclave2`, `myEnclave3`.

1. Measure all the enclaves:

```
sgx_emmt myApp.exe app_arg1 app_arg2
```

2. Measure two enclave targets:

```
sgx_ emmt --enclave=myEnclave1.signed.dll,myEnclave2.signed.dll myApp.exe
app_arg1 app_arg2
```

NOTE:

The enclave memory measurement tool works based on the assumption that the measurement targets are measurable enclaves and the symbol files of the measurement targets can be found by default. A measurable enclave should meet the following requirements:

1. The enclave should be a debuggable enclave. This means that the `<DisableDebug>` configuration parameter in the enclave configuration file should be set to 0.
2. This tool requires the enclave debug information. The enclave module should generate the debug information (`/Zi/ZI/Z7` and `/DEBUG`) at build time.
3. The enclave should be launched in debug mode. To launch the enclave in debug mode, set the debug flag to 1 when calling sgx_create_enclave to load the enclave.

NOTE:

Two versions of `sgx_emmt` are provided in the Intel® Software Guard Extensions Evaluation SDK: 32bit version and 64bit version. Cross utilizing the tool will cause a measurement failure. By default, the 64bit version is utilized. To measure 32bit enclaves, use the 32bit version `sgx_emmt` manually.

NOTE:

To enable the symbol files to be found by default, locate the symbol files where they are generated or place the symbol files at current working directory.

CPUSVN Configuration Tool

CPUSVN stands for Security Version Number of the CPU, which affects the key derivation and report generation process. CPUSVN is not a numeric concept and will be upgraded/downgraded along with the hardware upgrade/downgrade. To simulate the CPUSVN upgrade/downgrade without modifying the hardware, the Intel® Software Guard Extensions Evaluation SDK provides a CPUSVN configuration tool for you to configure the CPUSVN. The CPUSVN configuration tool is for Intel® SGX simulation mode only and can be launched as a command line tool or as a GUI tool. It depends on your input.

Command-Line Syntax

To run the Intel® SGX CPUSVN configuration tool, use the following syntax:

```
sgx_config_cpusvn [Command]
```

The valid commands are listed in the table below:

Table 6 CPUSVN Configuration Tool Commands

Command	Description
`-upgrade`	Simulate a CPUSVN upgrade.

`-downgrade`	Simulate a CPUSVN downgrade.
`-reset`	Restore the CPUSVN to its default value.

If the `[Command]` is omitted, the tool will be launched as a GUI tool and the following dialog will be shown. Then, you can simulate the CPUSVN upgrade/downgrade/reset by clicking the corresponding button.

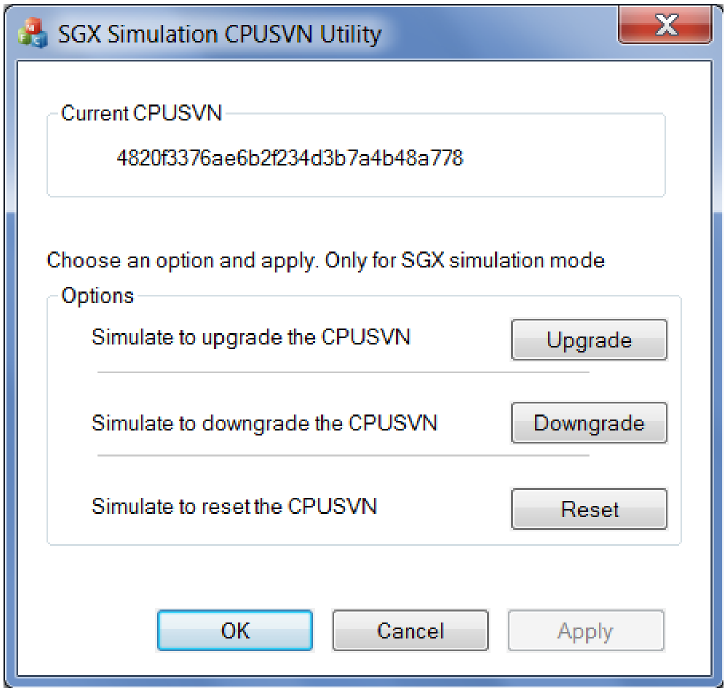

Figure 14 CPUSVN Configuration Tool Dialog

Enclave Development Basics

This topic introduces the following enclave development basics:

- Writing Enclave Functions
- Calling Functions inside the Enclave
- Calling Functions outside the Enclave
- Linking Enclave with Libraries
- Linking Application with Untrusted Libraries
- Enclave Definition Language Syntax
- Load and Unload an Enclave

The typical enclave development process includes the following steps:

1. Use IDE plug-in wizard to generate an enclave project. See Using Microsoft* Visual Studio* Intel® Software Guard Extensions Wizard for additional details.
2. Define the interface between the untrusted application and the enclave in the EDL file.
3. Implement the application and enclave functions.
4. Build the application and enclave. In the build process, The Edger8r Tool generates trusted and untrusted proxy/bridge functions. The Enclave Signing Tool generates the metadata and signature for the enclave.
5. Run and debug the application in simulation and hardware modes. See Enclave Debugger for more details.
6. Prepare the application and enclave for release.

Writing Enclave Functions

From an application perspective, making an enclave call (ECALL) appears as any other function call when using the untrusted proxy function. Enclave functions are plain C/C++ functions with several limitations.

The user can write enclave functions in C and C++ (native only). Other languages are not supported.

Enclave functions can rely on special versions of the C/C++ runtime libraries, STL, synchronization and several other trusted libraries that are part of the Intel® Software Guard Extensions Evaluation SDK. These trusted libraries are specifically designed to be used inside enclaves.

The user can write or use other trusted libraries, making sure the libraries follow the same rules as the internal enclave functions:

1. Enclave functions can’t use all the available 32-bit or 64-bit instructions. See Unsupported Instructions within an Enclave for a list of unsupported CPU instructions.
2. Enclave functions will only run in user mode (ring 3). Using instructions requiring other CPU privileges will cause the enclave to fault.
3. Function calls within an enclave are possible if the called function is statically linked to the enclave (the function needs to be in the enclave image file). Windows* Dynamic libraries are not supported.

CAUTION:

The enclave signing process will fail if the enclave image contains any unresolved dependencies at build time.

Calling functions outside the enclave is possible using what are called OCALLs. OCALLs are explained in detail in the Calling Functions outside the Enclave section.

Table 7 Summary of Intel® SGX Rules and Limitations

Feature	Supported	Comment
Languages	Partially	Native C/C++. Enclave interface functions are limited to C (no C++).
C/C++ calls to other DLLs	No	Can be done by explicit external calls (OCALLs).
C/C++ calls to System provided C/C++/STL standard libraries	No	A trusted version of these libraries is supplied with the Intel® Software Guard Extensions Evaluation SDK and they can be used instead.
OS API calls (e.g. WIN32)	No	Can be done by explicit external calls (OCALLs).
C++ frameworks	No	Including MFC*, QT*, Boost* (partially – as long as Boost runtime is not used).
Call C++ class methods	Yes	Including C++ classes, static and inline functions.
Intrinsic functions	Partially	Supported only if they use supported instructions. The allowed functions are included in the Intel® Software Guard Extensions Evaluation SDK.
Inline assembly	Partially	Same as the intrinsic functions.
Template functions	Partially	Only supported in enclave internal functions
Ellipse (...)	Partially	Only supported in enclave internal functions
Varargs (va_list)	Partially	Only supported in enclave internal functions.
Synchronization	Partially	The Intel® Software Guard Extensions Evaluation SDK provides a collection of functions/objects for synchronization: spin-lock, mutex, and condition variable.
Threading support	Partially	Creating threads inside the enclave is not supported. Threads that run inside the enclave are created within

		the (untrusted) application. Spin-locks, trusted mutex and condition variables API can be used for Thread Synchronization Primitives.
Thread Local Storage (TLS)	Partially	Only implicitly via declspec(thread). No dynamic allocation of TLS.
Dynamic memory allocation	Yes	Enclave memory is a limited resource. Maximum heap size is set at enclave creation.
C++ Exceptions	Yes	Although they have an impact on performance.
SEH Exceptions	No	The Intel® Software Guard Extensions Evaluation SDK supports a vector exception handling (VEH) like exception handling mechanism, see Custom Exception Handler for CPUID Instruction.

Calling Functions inside the Enclave

After an enclave is loaded successfully, you get an enclave ID which is provided as a parameter when the ECALLs are performed. Optionally, OCALLs can be performed within an ECALL. For example, assume that you need to compute some secret inside an enclave, the EDL file might look like the following:

```
// demo.edl
enclave {
     trusted {
          public void get_secret([out] secret_t* secret);
     };
     untrusted {
          // This OCALL is for illustration purposes only.
          // It should not be used in a real enclave, // unless it is dur-
          ing the development phase
          // for debugging purposes.
          void dump_secret([in] const secret_t* secret);
     };
};
```

With the above EDL, the sgx_edger8r will generate an untrusted proxy function for the ECALL and a trusted proxy function for the OCALL:

Untrusted proxy function:

```
sgx_status_t get_secret(sgx_enclave_id_t eid, secret_t* secret); // used by
the application
```

Trusted proxy function:

```
sgx_status_t dump_secret(const secret_t* secret); // used by the trusted
functions
```

The generated untrusted proxy function will automatically call into the enclave with the parameters to be passed to the real trusted function `get_secret` inside the enclave. To initiate an ECALL in the application:

```
// An enclave call (ECALL) will happen here
secret_t secret;
sgx_status_t status = get_secret(eid, &secret);
```

The trusted functions inside the enclave can optionally do an OCALL to dump the secret with the trusted proxy `dump_secret`. It will automatically call out of the enclave with the given parameters to be received by the real untrusted function `dump_secret`. The real untrusted function needs to be implemented by the developer and linked with the application.

Checking the Return Value

The trusted and untrusted proxy functions return a value of type `sgx_status_t`. If the proxy function runs successfully, it will return `SGX_SUCCESS`. Otherwise, it indicates a specific error described in Error Codes section. You can refer to the sample code shipped with the SDK for examples of proper error handling.

Calling Functions outside the Enclave

In exceptional cases, the code within the enclave needs to call external functions which reside in untrusted (unprotected) memory. This type of function call is named an OCALL.

These functions need to be declared in the EDL file in the untrusted section. See Enclave Definition Language Syntax for more details.

The enclave image is loaded very similarly to how Linux* OS loads shared objects. The function name space of the application is shared with the enclave so the enclave code can indirectly call functions linked with the application that created the enclave. Calling functions from the application directly is not permitted and will raise an exception at runtime.

CAUTION:

The wrapper functions copy the parameters from protected (enclave) memory to unprotected memory as the external function cannot access protected memory regions. In particular, the OCALL parameters are copied into the untrusted stack. Depending on the number of parameters, the OCALL may cause a stack overrun in the untrusted domain. The exception that this event will trigger will appear to come from the code that the sgx_eder8r generates based on the enclave EDL file. However, the exception can be easily detected using the Intel® SGX debugger. Accessing protected memory from unprotected memory will result in abort page semantics. This applies to all parts of the protected memory including heap, stack, code and data.

CAUTION:

Accessing protected memory from unprotected memory will result in abort page semantics. This applies to all parts of the protected memory including heap, stack, code and data.

The wrapper functions will copy buffers (memory referenced by pointers) only if these pointers are assigned special attributes in the EDL file.

CAUTION:

Certain trusted libraries distributed with the Intel® Software Guard Extensions Evaluation SDK provide an API that internally makes OCALLs. Currently, the Intel® Software Guard Extensions mutex, condition variable, and CPUID APIs from sgx_tstdc.lib make OCALLs. Similarly, the trusted support library sgx_tservice.lib, which provides services from the Platform Services Enclave (PSE-Op), also makes OCALLs. Developers who use these APIs must first import the needed OCALL functions from their corresponding EDL files. Otherwise, developers will get a linker error when the enclave is built. See the Enclave Definition Language Libraries - Creating a Trusted Library with Import/Export Functions for details on how to import OCALL functions from a trusted library EDL file.

CAUTION:

To help identify problems caused by missing imports, all OCALL functions used in the Intel® Software Guard Extensions Evaluation SDK have the suffix `ocall`. For instance, the linker error below indicates that the enclave needs to import the OCALLs `sgx_thread_wait_untrusted_event_ocall()` and `sgx_thread_set_untrusted_event_ocall()` are needed in `sethread_mutex.obj`, which is part of `sgx_tstdc.lib`.

```
sgx_tstdc.lib(sethread_mutex.obj) : error LNK2001: unresolved external
symbol _sgx_thread_wait_untrusted_event_ocall
sgx_tstdc.lib(sethread_mutex.obj) : error LNK2001: unresolved external
symbol _sgx_thread_set_untrusted_event_ocall
```

CAUTION:

Abort page semantics:

An attempt to read from a non-existent or disallowed resource returns all ones for data (abort page). An attempt to write to a non-existent or disallowed physical resource is dropped. This behavior is unrelated to exception type abort (the others being Fault and Trap).

OCALL functions have the following limitations/rules:

- OCALL functions must be C functions, or C++ functions with C linkage.
- Pointers that reference data within the enclave must be annotated with pointer direction attributes in the EDL file. The wrapper function will perform shallow copy on these pointers. See Pointers for more information.
- Exceptions will not be caught within the enclave. The user must handle this in the untrusted wrapper function.
- OCALLs cannot have an ellipse (...) or a `va_list` in their prototype.

Example 1: The definition of a simple OCALL function

```
// foo.edl
enclave {
    untrusted {
        [cdecl] void foo(int param);
    };
};
```

Step 1 – Add a declaration for `foo` in the EDL file

Step 2 (optional but highly recommended) – a write trusted, user-friendly wrapper.

This function is part of the enclave's trusted code.

The wrapper function `ocall_foo` function will look like:

```
void ocall_foo(int param)
{
     // it is necessary to check the return value of foo()
     if (foo(param) != SGX_SUCCESS)
          abort();
}
```

Step 3 – write untrusted `foo` function

The `sgx_edger8r` will generate an untrusted bridge function which will call the real untrusted function `foo` automatically. This untrusted bridge and the target untrusted function are part of the application, not the enclave.

Library Development for Enclaves

Trusted library is the term used to refer to a static library designed to be linked with an enclave. The following list describes the features of trusted libraries:

- Trusted libraries are components of an Intel® SGX-based solution. They typically undergo a more rigorous threat evaluation and review process than a regular static library.
- A trusted library is developed (or ported) with the specific purpose of being used within an enclave. Therefore, it should not contain instructions that are not supported by the Intel® SGX architecture..
- A subset of the trusted library API may also be part of the enclave interface. The trusted library interface that could be exposed to the untrusted domain is defined in an EDL file. If present, this EDL file is a key component of the trusted library.
- A trusted library may have to be shipped with an untrusted library. Functions within the trusted library may make OCALLs outside the enclave. If an external function that the trusted library uses is not provided by the libraries available on the platform, the trusted library will require an untrusted support library.

In summary, a trusted library, in addition to the `.lib` file containing the trusted code and data, may also include an `.edl` file as well as an untrusted `.lib` file.

This topic describes the process of developing a trusted library and provides an overview of the main steps necessary to build an enclave that uses such a trusted library.

1. The ISV provides a trusted library including the trusted functions (without any edge-routines) and, when necessary, an EDL file and an untrusted support library. To develop a trusted library, an ISV should create an enclave project and choose the library option in the Intel® SGX Wizard. This ensures the library is built with the appropriate settings. The ISV might delete the EDL file from the project if the trusted library only provides an interface to be invoked within an enclave. The ISV should create a standard static library project for the untrusted support library, if required.
2. Add a "from/import" statement with the library EDL file path and name to the enclave EDL file. The import statement indicates which trusted functions (ECALLs) from the library may

be called from outside the enclave and which untrusted functions (OCALLs) are called from within the trusted library. You may import all ECALLs and OCALLs from the trusted library or select a specific subset of them.
A library EDL file may import additional library EDL files building a hierarchical structure. For additional details, See Enclave Definition Language Libraries - Creating a Trusted Library with Import/Export Functions.

3. During the enclave build process, the `sgx_edger8r` generates proxy/bridge code for all the trusted and untrusted functions. The generated code accounts for the functions declared in the enclave EDL file as well as any imported trusted library EDL file.
4. The trusted library and trusted proxy/bridge functions are linked to the enclave code.

NOTE:

If you use the wildcard option to import a trusted library, the resulting enclave contains the trusted bridge functions for all ECALLs and their corresponding implementations. The linker will not be able to optimize this code out.

---

5. The Intel® SGX application is linked to the untrusted proxy/bridge code. Similarly, when the wildcard import option is used, the untrusted bridge functions for all the OCALLs will be linked in.

Avoiding Name Collisions

An application may be designed to work with multiple enclaves. In this scenario, each enclave would still be an independent compilation unit resulting in a separate DLL file.

Enclaves, like regular DLL files, should provide a unique interface to avoid name collisions when an untrusted application is linked with the edge-routines of several enclaves. The `sgx_edger8r` prevents name collisions among OCALL functions because if automatically prepends the enclave name to the names of the untrusted bridge functions. However, ISVs must ensure the uniqueness of the ECALL function names across enclaves to prevent collisions among ECALL functions.

Despite having unique ECALL function names, name collision may also occur as the result of developing an Intel® SGX application. This happens because an enclave cannot import another DLL file . When two enclaves import the same ECALL function from a trusted library, the set of edge-routines for each enclave will contain identical untrusted proxy functions and marshaling data structures for the imported ECALL. Thus, the linker will emit an error when the application is linked with these two sets of edge-routines. To build an application with more than one enclave when these enclaves import the same ECALL from a trusted library, ISVs have to:

1. Provide the `--use-prefix` option to sgx_edger8r, which will prepend the enclave name to the untrusted proxy function names.For instance, when an enclave uses the local attestation trusted library sample code included in the Intel® SGX Evaluation SDK, the enclave EDL file must be parsed with the `--use-prefix` sgx_edger8r option. See Local Attestation for additional details.
2. Prefix all ECALLs in their untrusted code with the enclave name, matching the new proxy function names.

Linking Enclave with Libraries

This topic introduces how to link an enclave with the following types of libraries:

- Dynamic libraries
- Static Libraries
- Simulation Libraries

Dynamic Libraries

An enclave DLL must *not* depend on any dynamically linked library in any way. The enclave loader has been intentionally designed to prohibit dynamic linking of libraries within an enclave. The protection of an enclave is dependent upon obtaining an accurate measurement of all code and data that is placed into the enclave at load time; thus, dynamic linking would add complexity without providing any benefit over static linking.

CAUTION:

The enclave image signing process will fail if the enclave file has any unresolved dependencies. It means that a DLL must have an empty IAT (Import Address Table).

Static Libraries

The user can link with static libraries as long as they do not have any dependencies.

The Intel® Software Guard Extensions Evaluation SDK provides the following collection of trusted libraries.

Table 8 Trusted Libraries included in the Intel® SGX Evaluation SDK

Name	Description	Comment
`sgx_trts.lib`	Intel® SGX internals	Must link when running in HW mode
`sgx_trts_sim.lib`	Intel® SGX internals (simulation mode)	Must link when running in simulation mode
`sgx_tstdc.lib`	Standard C library (math, string, etc.)	Must link
`sgx_tstdcxx.lib`	Standard C++ libraries, STL	Optional
`sgx_tservice.lib`	Data seal/unseal (encryption), trusted Architectural Enclaves support, Elliptic Curve Diffie-Hellman (EC DH) library, etc.	Must link when using HW mode
`sgx_tservice_sim.lib`	The counterpart of sgx_tservice.lib for simulation mode	Must link when using simulation mode
`sgx_tcrypto.lib`	Cryptographic library	You must choose one cryptographic library to link. Recommendation setting is to link the optimized version when using hardware mode and use the other

		version for simulation mode.
`sgx_tcrypto_opt.lib`	Optimized Cryptographic library, size-wise.	
`sgx_tkey_exchange.lib`	Trusted key exchange library	Optional

Simulation Libraries

The Intel® SGX Evaluation SDK provides simulation libraries to run application enclaves in simulation mode (Intel® SGX hardware is not required). There are an untrusted simulation library and a trusted simulation library. The untrusted simulation library provides the functionality that the untrusted runtime library requires to manage an enclave linked with the trusted simulation library, including the simulation of the Intel® SGX instructions executed outside the enclave: ECREATE, EADD, EEXTEND, EINIT, EREMOVE, and ECREATE. The trusted simulation library is primarily responsible for simulating the Intel® SGX instructions that can executed inside an enclave: EEXIT, EGETKEY, and EREPORT.

Linking Application with Untrusted Libraries

The Intel® Software Guard Extensions Evaluation SDK provides the following collection of untrusted libraries.

Table 9 Untrusted Libraries included in the Intel® SGX Evaluation SDK

Name	Description	Comment
`sgx_urts.lib`	Provides functionality for applications to manage enclaves	Must link when running in HW mode. `sgx_urts.dll` is included in Intel® SGX PSW
`sgx_urts_sim.dll`	uRTS library used in simulation mode	Dynamic linked
`sgx_urts_sim.lib`	The counterpart of `sgx_urts.lib` for simulation mode	Must link when running in simulation mode
`sgx_uae_service.lib`	Provides both enclaves and untrusted applications access to services provided by the AEs	Must link when running in HW mode. `sgx_uae_service.dll` is included in Intel® SGX PSW
`sgx_uae_service_sim.dll`	Untrusted AE support library used in simulation mode	Dynamic linked
`sgx_uae_service_`	The counterpart of `sgx_uae_service.lib`	Must link when run-

sim.lib	for simulation mode	ning in simulation mode
sgx_ukey_ exchange.lib	Untrusted key exchange library built with /MD	Optional
sgx_ukey_ exchangemt.lib	Untrusted key exchange library built with /MT	Optional
sgx_status.dll	Provides functionality for applications to register Enclave Signing Key White List Certificate Chain	Optional
sgx_capable.dll	Provides functionality for applications to check if the client platform is enabled for Intel SGX or to enable the Intel SGX device	Optional

Enclave Definition Language Syntax

Enclave Definition Language (EDL) files are meant to describe enclave trusted and untrusted functions and types used in the function prototypes. The Edger8r Tool uses this file to create C wrapper functions for both enclave exports (used by ECALLs) and imports (used by OCALLs).

EDL Template

```
enclave {
    //Include files

    //Import other edl files

    //Data structure declarations to be used as parameters of the
    //function prototypes in edl

    trusted {
        //Include file if any.
        //It will be inserted in the trusted header file (enclave_t.h)

        //Trusted function prototypes

    };

    untrusted {
        //Include file if any.
        //It will be inserted in the untrusted header file (enclave_u.h)

        //Untrusted function prototypes

    };
};
```

The trusted block is optional only if it is used as a library EDL, and this EDL would be imported by other EDL files. However the untrusted block is always optional.

Every EDL file follows this generic format:

```
enclave {
     // An EDL file can optionally import functions from other EDL files.
     from "other/file.edl" import foo, bar;  // selective importing
     from "another/file.edl" import *;       // import all functions

     // Include C headers, these headers will be included in the generated
     files.
     // for both trusted and untrusted routines.
     include "string.h"
     include "mytypes.h"

     // Type definitions (struct, union, enum), optional.
     struct mysecret {
          int key;
          const char* text;
     };
     enum boolean { FALSE = 0, TRUE = 1 };

     // Export functions (ECALLs), optional for library EDLs.
     trusted {
          //Include file if any.
          //It will be inserted in trusted header file
          include "trusted.h"

          //Trusted function prototypes

          public void set_secret([in] struct mysecret* psecret);

          void some_private_func(enum boolean b); // private ECALL (non-
          root ECALL).
     };

     // Import functions (OCALLs), optional.
     untrusted {

          //Include file if any.
          //It will be inserted in untrusted header file
          include "untrusted.h"

          //Untrusted function prototypes

          // This OCALL is not allowed to make another ECALL.
          void ocall_print();

          // This OCALL can make an ECALL to function "some_private_func".
          int another_ocall([in] struct mysecret* psecret)
               allow(some_private_func);
     };
};
```

Comments

Both types of C/C++ comments are valid.

Example

```
enclave {
```

```
    include "stdio.h" // include stdio header
    include "../../util.h" /* this header defines some custom public types
    */
};
```

Include Headers

Include C headers which define types (C structs, unions, typedefs, etc.); otherwise auto generated code cannot be compiled if these types are referenced in EDL. The included header file can be global or belong to trusted functions or untrusted functions only.

A global included header file doesn't mean that the same header file is included in the enclave and untrusted application code. In this case, the enclave will use the `stdio.h` from the Intel® Software Guard Extensions Evaluation SDK. While the application code will use the `stdio.h` shipped with the host compiler.

Using the `include` directive is convenient when developers are migrating existing code to enclave technology, since data types are defined already in this case. Similar to other IDL languages like Microsoft* interface definition language (MIDL*) and CORBA* interface definition language (OMG-IDL), also supported is that a user can define data types inside the EDL file and `sgx_edger8r` will generate a C header file with the data type definitions. For a list of supported data types with in EDL, see Basic Types.

Syntax

```
include "filename.h"
```

Example

```
enclave {
    include "stdio.h"       // global headers
    include "../../util.h"

    trusted {
        include "foo.h"    // for trusted functions only
    };

    untrusted {
        include "bar.h"    // for untrusted functions only
    };
};
```

Keywords

The identifiers listed in the following table are reserved for use as keywords of the Enclave Definition Language.

Table 10 EDL Reserved Keywords

Data Types					
char	short	int	float	double	void
int8_t	int16_t	int32_t	int64_t	size_t	wchar_t

uint8_t	uint16_t	uint32_t	uint64_t	unsigned	struct
union	enum	long			
Pointer Parameter Handling					
in	out	user_ check	count	size	readonly
isptr	sizefunc	string	wstring		
Others					
enclave	from	import	trusted	untrusted	include
public	allow	isary	const		
Function Calling Convention					
cdecl	stdcall	fastcall	dllimport		

Basic Types

EDL supports the following basic types:

```
char, short, long, int, float, double, void, int8_t, int16_t, int32_t,
int64_t, size_t, wchar_t, uint8_t, uint16_t, uint32_t, uint64_t, unsigned,
struct, enum, union.
```

It also supports `long long` and `long double`.

Basic data types can be modified using the C modifiers:

```
const, *, [].
```

Additional types can be defined by including a C header file.

Structures, Enums and Unions

Basic types and user defined data types can be used inside the structure/union except it differs from the standard in the following ways:

Illegal Syntax:

```
enclave{
     // 1. Each member of the structure has to be
     // defined separately
     struct data_def_t{
          int a, b, c; // Not allowed
                       // It has to be int a; int b; int c;
     };

     // 2. Bit fields in structures/unions are not allowed.
     struct bitfields_t{
          short i : 3;
          short j : 6;
          short k : 7;
     };
```

```
    //3. Nested structure definition not allowed
    struct my_struct_t{
        int out_val;
        float out_fval;
        struct inner_struct_t{
            int in_val;
            float in_fval;
        };
    };
};
```

Valid Syntax:

```
enclave{

    include "user_types.h" //for ufloat: typedef float ufloat

    struct struct_foo_t {
        uint32_t struct_foo_0;
        uint64_t struct_foo_1;
    };

    enum enum_foo_t {
        ENUM_FOO_0 = 0,
        ENUM_FOO_1 = 1
    };

    union union_foo_t {
        uint32_t union_foo_0;
        uint32_t union_foo_1;
        uint64_t union_foo_3;
    };

    trusted {

        public void test_char(char val);
        public void test_int(int val);
        public void test_long(long long val);

        public void test_float(float val);
        public void test_ufloat(ufloat val);
        public void test_double(double val);
        public void test_long_double(long double val);

        public void test_size_t(size_t val);
        public void test_wchar_t(wchar_t val);

        public void test_struct(struct struct_foo_t val);
        public void test_struct2(struct_foo_t val);

        public void test_enum(enum enum_foo_t val);
        public void test_enum2(enum_foo_t val);

        public void test_union(union union_foot_t val);
        public void test_union2(union_foo_t val);
    };
```

```
};
```

Pointers

EDL defines several attributes that can be used with pointers:

`in, out, user_check, string, wstring, size, count, sizefunc, isptr, readonly.`

Each of them is explained in the following topics.

CAUTION:

The pointer attributes explained in this topic apply to ECALL and OCALL function parameters exclusively, not to the pointers returned by an ECALL or OCALL function. Thus, pointers returned by an ECALL or OCALL function are not checked by the edge-routines and must be verified by the enclave code.

Pointer Handling

The [in], [out] and [user_check] are used for handling pointers. The [in] and [out] serve as direction attributes.

- [in] – when [in] is specified for a pointer argument, the parameter is passed from the calling procedure to the called procedure. For an ECALL the `in` parameter is passed from the application to the enclave, for an OCALL the parameter is passed from the enclave to the application.
- [out] – when [out] is specified for a pointer argument, the parameter is returned from the called procedure to the calling procedure. In an ECALL function an `out` parameter is passed from the enclave to the application and an OCALL function passes it from the application to the enclave.
- [in] and [out] attributes may be combined. In this case the parameter is passed in both directions.

The direction attribute instructs the trusted edge-routines (trusted bridge and trusted proxy) to copy the buffer pointed by the pointer. In order to copy the buffer contents, the trusted edge-routines have to know how much data needs to be copied. For this reason, the direction attribute is usually followed by a `size`, `count` or `sizefunc` modifier. If neither of these are provided is provided nor the pointer is NULL, the trusted edge-routine assumes a `count` of one. When a buffer is being copied, the trusted bridge must avoid overwriting enclave memory in an ECALL and the trusted proxy must avoid leaking secrets in an OCall. To accomplish this goal, pointers passed as ECALL parameters must point to untrusted memory and pointers passed as OCALL parameters must point to trusted memory. If these conditions are not satisfied, the trusted bridge and the trusted proxy will report an error, respectively, and the ECALL and OCALL functions will not be executed.

In ECALLs, the trusted bridge checks that the marshaling structure does not overlap enclave memory, and automatically allocates space on the trusted stack to hold a copy of the structure. Then it checks that pointer parameters with their full range do not overlap with enclave memory. When a pointer to untrusted memory with attribute `in` is passed to the enclave, the trusted bridge allocates memory inside the enclave and copies the memory pointed to by the pointer from outside to the enclave memory. When a pointer to untrusted memory with the `out` attribute is passed to the enclave, the trusted bridge allocates a buffer in trusted memory, zeroes the buffer contents to clear any previous secrets and passes a pointer to this buffer to the trusted function. After the trusted function returns, the trusted bridge copies the contents of the trusted buffer to untrusted memory. When the `in` and `out` attributes are combined, the trusted bridge allocates memory inside the enclave, makes a copy of the buffer in trusted memory before calling the trusted function, and once the trusted function returns, the trusted bridge copies the contents of the trusted buffer to untrusted memory. The amount of data copied out is the same as the amount of data copied in.

NOTE:

Due to the fact that the `sgx_edger8r` tool does not know how to check the return value of the real trusted function, the generated code will always copy the buffer outside the enclave when the buffer corresponds to an ECALL pointer parameter declared with the "out" attribute. You must clear all sensitive data from that buffer on failure.

For OCALLs, the trusted proxy allocates memory on the outside stack to pass the marshaling structure and checks that pointer parameters with their full range are within enclave. When a pointer to trusted memory with attribute `in` is passed from an enclave (an OCALL), the trusted proxy allocates memory outside the enclave and copies the memory pointed by the pointer from inside the enclave to untrusted memory. When a pointer to trusted memory with the `out` attribute is passed from an enclave (an OCALL), the trusted proxy allocates a buffer on the untrusted stack, and passes a pointer to this buffer to the untrusted function. After the untrusted function returns, the trusted proxy copies the contents of the untrusted buffer to trusted memory. When the `in` and `out` attributes are combined, the trusted proxy allocates memory outside the enclave, makes a copy of the buffer in untrusted memory before calling the untrusted function, and after the untrusted function returns the trusted proxy copies the contents of the untrusted buffer to trusted memory. The amount of data copied out is the same as the amount of data copied in.

Before the trusted bridge returns, it frees all the trusted heap memory allocated at the beginning of the ECALL function for pointer parameters with a direction attribute. Similarly, when the trusted proxy function returns, it frees all the untrusted stack memory allocated at the beginning of the OCALL function for pointer parameters with a direction attribute. Attempting to use a buffer allocated by the trusted bridge or trusted proxy after these functions return will result in undefined behavior.

You may use the direction attribute to trade protection for performance. Otherwise, you must use the `user_check` attribute described below and validate the data obtained from untrusted memory via pointers before using it, since the memory a pointer points to could change unexpectedly because it is stored in untrusted memory. However, the direction attribute does not help with structures that contain pointers. In this scenario, developers have to validate and copy the buffer contents, recursively if needed, themselves.

User Check Attribute

In certain situations, the restrictions imposed by the direction attribute may not support the application needs for data communication across the enclave boundary. For instance, a buffer might be too large to fit in enclave memory and needs to be fragmented into smaller blocks that are then processed in a series of ECALLs, or an application might require passing a pointer to trusted memory (enclave context) as en ECALL parameter.

To support these specific scenarios, the EDL language provides the `user_check` attribute. Parameters declared with the `user_check` attribute do not undergo any of the checks described for `[in]` and `[out]` attributes. However, the ISV must understand the risks associated with passing pointers in and out the enclave, in general, and the `user_check` attribute, in particular. The ISV must ensure that all the pointer checking and data copying is done correctly or risk compromising enclave secrets.

Example

```
enclave {

     trusted {

          public void test_ecall_user_check([user_check] int * ptr);

          public void test_ecall_in([in] int * ptr);

          public void test_ecall_out([out] int * ptr);

          public void test_ecall_in_out([in, out] int * ptr);

     };

     untrusted {

          void test_ocall_user_check([user_check] int * ptr);

          void test_ocall_in([in] int * ptr);

          void test_ocall_out([out] int * ptr);

          void test_ocall_in_out([in, out] int * ptr);

     };

};
```

Illegal Syntax:

```
enclave {

     trusted {

          // 1.Pointers without any direction attributes
          // or 'user_check' are not allowed in edl.

          public void test_ecall_not(int * ptr);

          // 2.Function pointers are not allowed

          public void test_ecall_func([in]int (*func_ptr)());

     };
};
```

In the example shown above:

For ECALL:

- [user_check]: In the function `test_ecall_user_check`, the pointer `ptr` will not be verified; you should verify if the pointer has been passed to the trusted function. The buffer pointed to by `ptr` is not copied to inside buffer either.
- [in]: In the function `test_ecall_in`, a buffer with the same size as the data type of 'ptr'(int) will be allocated inside the enclave. Content pointed to by `ptr`, one integer value, will be copied

into the new allocated memory inside. Any changes performed inside the enclave will not be visible to the untrusted application.

- [out]: In the function `test_ecall_out`, a buffer with the same size as the data type of 'ptr' (int) will be allocated inside the enclave, but the content pointed to by `ptr`, one integer value will not be copied. Instead, it will be initialized to zero. After the trusted function returns, the buffer inside the enclave will be copied to the outside buffer pointed to by `ptr`.
- [in, out]: In the function `test_ecall_in_out`, a buffer with the same size will be allocated inside the enclave, the content pointed to by `ptr`, one integer value, will be copied to this buffer. After returning, the buffer inside the enclave will be copied to the outside buffer.

For OCALL:

- [user_check]: In the function `test_ocall_user_check`, the pointer `ptr` will not be verified; the buffer pointed to by `ptr` is not copied to an outside buffer. Besides, the application cannot read/modify the memory pointed to by `ptr`, if `ptr` points to enclave memory.
- [in]: In the function `test_ocall_in`, a buffer with the same size as the data type of `ptr`(int) will be allocated in 'application' side (untrusted side). Content pointed to by `ptr` will be copied into the newly allocated memory outside. Any changes performed by the applicationwill not be visible inside the enclave.
- [out]: In the function `test_ocall_out`, a buffer with the same size as the data type of `ptr` (int) will be allocated on theapplication side (untrusted side) and its content will be initialized to zero.After the untrusted function returns, the buffer outside the enclave will be copied to the enclave buffer pointed to by `ptr`.
- [in, out]: In the function `test_ocall_in_out`, a buffer with the same size will be allocated in the application side, the content pointed to by `ptr` will be copied to this buffer. After returning, the buffer outside the enclave will be copied into the inside enclave buffer.

The following table summarizes wrapper function behaviors when using the in/out attributes:

Table 11 wrapper function behaviors when using the in/out attributes

	ECALL	OCALL
user_check	Pointer is not checked. Users must perform the check and/or copy.	Pointer is not checked. Users must perform the check and/or copy
in	Buffer copied from the application into the enclave. Afterwards, changes will only affect the buffer inside enclave. Safe but slow.	Buffer copied from the enclave to the application. Must be used if pointer points to enclave data.
out	Trusted wrapper function will allocate a buffer to be used by the enclave. Upon return, this buffer will be copied to the original buffer.	The untrusted buffer will be copied into the enclave by the trusted wrapper function. Safe but slow.
in, out	Combines `in` and `out` behavior. Data is copied back and forth.	Same as ECALLs.

Attributes for Buffer Size Calculation

The generalized formula for calculating the buffer size using these attributes:

```
Total number of bytes = count * size
```

- The above formula holds when both `count` and `size/sizefunc` are specified.
- `size`can be specified by either `size` or `sizefunc` attribute.
- If count is not specified for the pointer parameter, then it is assumed to be equal to `1`, for example, `count=1`. Then total number of bytes equals to `size/sizefunc`.
- If `size` is not specified, then the buffer size is calculated using the above formula where `size` is *sizeof (element pointed by the pointer)*.

Attribute: size

The `size` attribute is used to indicate the buffer size in bytes used for copy depending on the direction attribute (`[in]`/`[out]`) (when there is no `count` attribute specified). This attribute is needed because the trusted bridge needs to know the whole range of the buffer passed as a pointer to ensure it does not overlap with the enclave memory, and to copy the contents of the buffer from untrusted memory to trusted memory and/or vice versa depending on the direction attribute. The size may be either an integer constant or one of the parameters to the function. `size` attribute is generally used for `void` pointers.

Attribute: count

Count attribute is used to indicate a block of `sizeof` element pointed by the pointer in bytes used for copy depending on the direction attribute. The `count` and `size` attribute modifiers serve the same purpose. The number of bytes copied by the trusted bridge or trusted proxy is the product of the count and the size of the data type to which the parameter points. The count may be either an integer constant or one of the parameters to the function.

The `size` and `count` attribute modifiers may also be combined. In this case, the trusted edge-routine will copy a number of bytes that is the product of the count and size parameters (size*-count) specified in the function declaration in the EDL file.

Attribute: sizefunc

The `sizefunc` attribute modifier depends on a user defined trusted function which is called by the edge-routines to get the number of bytes to be copied. The `sizefunc` has similar functionality as the `sizeof()` operator. An example of where `sizefunc` can be used is for marshaling variable-length structures, which are buffers whose total size is specified by a combination of values stored at well-defined locations inside the buffer (although typically it is at a single location). To prevent "check first, use later" type of attacks, `sizefunc` is called twice. In the first call, `sizefunc` operates in untrusted memory. The second time, `sizefunc` operates in the data copied into trusted memory. If the sizes returned by the two `sizefunc` calls do not match, the trusted bridge will cancel the ECALL and will report an error to the untrusted application. Note that `sizefunc` must not be combined with the `size` attribute. `sizefunc` cannot be used with `out` alone, however `sizefunc` with both `in` and `out` is accepted. Additionally, users cannot define`sizefunc` as `strlen` or `wcslen`. In all these scenarios, the `sgx_edger8r` will throw an error. Strings should not be passed with the `sizefunc` modifier, but with the `string` or `wstring` keyword. `sizefunc` can be used with the `count` attribute which gives the total length to be equal to `sizefunc` * `count`. The following items are the prototype of the trusted `sizefunc` that you need to define inside the enclave:

```
size_t sizefunc_function_name(const parameter_type * p);
```

Where `parameter_type` is the data type of the parameter annotated with the `sizefunc` attribute. If you do not provide the definition of the `sizefunc` function, the linker will report an error.

NOTE

The function implementing a `sizefunc` should validate the input pointer carefully, before really using it. Since the function is called before the pointer is checked by the generated code.

Example

```
enclave{

    trusted {

        // Copies '100' bytes

        public void test_size1([in, size=100] void* ptr, size_t len);

        // Copies 'len' bytes

        public void test_size2([in, size=len] void* ptr, size_t len);

        // Copies cnt * sizeof(int) bytes

        public void test_count([in, count=cnt] int* ptr, unsigned cnt);

        // Copies cnt * len bytes

        public void test_count_size([in, count=cnt, size=len] int* ptr,
                        unsigned cnt, size_t len);

        // Copies get_packet_size bytes
        // User must provide a function definition that matches
        // size_t get_packet_size(const void* ptr);

        void test_sizefunc([in, sizefunc=get_packet_size] void* ptr);

        // Copies (get_packet_size * cnt) bytes

        void test_sizefunc2(
                        [in,  sizefunc=get_ packet_ size,  count=cnt]  void*
                        ptr,
                        unsigned cnt);
    };
};
```

Illegal Syntax:

```
enclave{

    include "user_types.h"

    trusted {

        // size/count/sizefunc attributes must be used with
        // pointer direction ([in, out])

        void test_attribute_cant([size=len] void* ptr, size_t len);

        // Cannot use sizefunc and size together
```

```
        void test_sizefunc_size(
                    [in, size=100, sizefunc=packet_len] header* h);

        // Cannot use strlen or wcslen as sizefunc

        void test_sizefunc_strlen([in, sizefunc=strlen] header* h);
        void test_sizefunc_wcslen([in, sizefunc=wcslen] header* h);
    };
};
```

Attribute: string/wstring

The attributes `string` and `wstring` indicate that the parameter is a NULL terminated C string or a NULL terminated `wchar_t` string, respectively. To prevent "check first, use later" type of attacks, the trusted edge-routine first operates in untrusted memory to determine the length of the string. Once the string has been copied into the enclave, then the trusted bridge explicitly NULL terminates the string. The size of the buffer allocated in trusted memory accounts for the length is determined in the first step as well as the size of the string termination character.

NOTE

The `string`and `wstring` attributes must not be combined with any other modifier such as `size`, `count` or `sizefunc`. `string` and `wstring` cannot be used with `out` alone, however, `string` and `wstring` with both `in` and `out` are accepted. In these cases, the `sgx_edger8r` will report an error.

Example

```
enclave {

    include "user_types.h"  // for typedef void * pBuf;
                             // and typedef void const * pBuf2;

    trusted {

        // Cannot use [out] with "string/wstring" alone
        // Using [in] , or [in, out] is acceptable

        public void test_string([in, out, string] char* str);

        public void test_wstring([in, out, wstring] char* wstr);

        public void test_const_string([in, string] const char* str);

        public void test_isptr(
                    [in, isptr, size=len] pBufptr,
                    size_t len);

        public void test_isptr_readonly(
                    [in, out, isptr, readonly, size=len] pBuf2ptr,
                    size_t len);

    };
};
```

Illegal Syntax:

```
enclave {

    include "user_types.h"  //for typedef void const * pBuf2;

    trusted {

         // string/wstring attributes must be used
         // with pointer direction

         void test_string_cant([string] char* ptr);

         // string/wstring attributes cannot be used
         // with [out] attribute

         void test_string_out([out, string] char* str);

         // sizefunc can't be used for strings, use [string/wstring]

         void test_string_sizefunc_cant(
                        [in, string, sizefunc=packet_len] header* h);

         // Cannot use [out] when using [readonly] attribute

         void test_isptr_readonly_cant(
                        [in, out, isptr, readonly, size=len] pBuf2ptr,
                        size_t len);
    };

};
```

In the above example, when the `string` attribute is used for function `test_string` , `strlen (str)+1` is used as the size for copying the string in and out of the enclave. The extra byte is for null termination.

In the function `test_wstring`, `wcslen(str)+1` (two-byte units) will be used as the size for copying the string in and out of the enclave.

In the function `test_isptr_readonly`, `pBuf2` (`typedef void const * pBuf2`) is a user defined pointer type, so `isptr` is used to indicate that it is a user defined type. Also, the `ptr` is `readonly`, so you cannot use the `out` attribute. The `size` attribute indicates the number of bytes to be copied to the enclave memory.

CAUTION:

Pointers should be decorated with either a pointer direction attribute `in`, `out` or a `user_check` attribute explicitly.

EDL cannot analyze C typedefs and macros found in C headers. If a pointer type is aliased to a type/macro that does not have an asterisk (*), the EDL parser may report an error or not properly copy the pointer's data.

In such cases, declare the function prototype to use types that have an asterisk.

Example:

```
void foo([in, size=4] PVOID buffer); // error, PVOID is not a pointer in
EDL
void foo([in, size=4] void* buffer); // OK
void foo([in, isptr, size=4] PVOID buffer);
                        // OK, "isptr" indicates "PVOID" is pointer type.
void foo(HWND hWnd);
                        // OK, opaque type, copy by value.
                        // Actual address must be in untrusted memory
```

User Defined Data Types

The Enclave Definition Language (EDL) supports user defined data types, but should be defined in a header file. Any basic datatype which is typedef'ed into another becomes a user defined data type.

Some user data types need to be annotated with special EDL attributes, such as `isptr`, `isary` and `readonly`, explained below. If one of these attributes is missing when a user-defined type parameter requires it so, the compiler will emit a compilation error in the code that `sgx_edger8r` generates.

When there is a user defined data type for pointer, `isptr` is used to indicate that the user defined parameter is a pointer. See Pointers for more information.

When there is a user defined data type for arrays, `isary` is used to indicate that the user defined parameter is an array. See Arrays for more information.

const Keyword and readonly Attribute

The EDL language accepts the `const` keyword with the same meaning as the `const` keyword in the C standard. However, the support for this keyword is limited in the EDL language. It may only be used with pointers and as the outermost qualifier. This satisfies the most important usage in Intel® SGX, which is to detect conflicts between const pointers (pointers to const data) with the `out` attribute. Other forms of the `const` keyword supported in the C standard are not supported in the EDL language.

When an ECALL or OCALL parameter is a user defined type of a pointer to a const data type, the parameter should be annotated with the `readonly` attribute.

Arrays

The Enclave Definition Language (EDL) supports multidimensional, fixed-size arrays to be used in data structure definition and parameter declaration. Zero-length array and flexible array member, however, are *not* supported. The special attribute `isary` is used to designate function parameters that are of a user defined type array.

Example

```
enclave {

    include "user_types.h"  //for uArray - typedef int uArray[10];

    trusted {

        public void test_array([in] int arr[4]);

        public void test_array_multi([in] int arr[4][4]);

        public void test_isary([in, isary, size=len] uArray arr,
```

```
                        size_tlen);
    };
};
```

Illegal Syntax:

```
enclave {

    include "user_types.h"  //for uArray - typedef int uArray[10];

    trusted {

        // Flexible array is not supported
        public void test_flexible(int arr[][4]);

        // Zero-length array is not supported.
        public void test_zero(int arr[0]);

        // User-defined array types need "isary"
        public void test_miss_isary([in, size=len] uArray arr,
                        size_t len);

    };
};
```

Support for arrays also includes attributes `[in]`, `[out]` and `[user_check]`, which are similar in usage to the pointers.

Function Calling Convention for OCALLs

Untrusted functions can optionally receive attributes that affect their calling convention and DLL linkage. You can find details on these calling conventions at http://msdn.microsoft.com/en-us/library/984x0h58

The `cdecl` calling convention is the default as defined by the C standard.

Improper use of the `cdecl`, `stdcall` or `fastcall` keywords may result in a linker error.

OCALL functions (untrusted) may be implemented in DLLs, the keyword `dllimport` is used to specify this attribute. Improper use of the `dllimport` keyword will result in a compilation warning.

The calling convention is specified using the following keywords:

Table 12 Calling Convention Keywords

Value	Stack Cleanup	Parameter Passing
cdecl	Caller	Pushes parameters on the stack (right to left)
stdcall	Callee	Pushes parameters on the stack (right to left)
fastcall	Callee	Stored in registers, then pushed on stack (right to left)

These calling conventions affect 32-bit builds only. 64-bit builds have a single calling convention, `fastcall`.

Example

The trusted function `test_calling_convs()` can use the standard functions like file operations and others by using untrusted functions(OCALLs).

```
enclave {

    trusted {
        public void test_calling_convs(void);
    };


    untrusted {

        [cdecl, dllimport] FILE * fopen(
                      [in,string] const char * filename,
                      [in,string] const char * mode);

        [cdecl, dllimport] int fclose([user_check] FILE * stream);

        [cdecl, dllimport] size_t fwrite(
                      [in, size=size, count=count] const void * buffer,
                      size_t size,
                      size_t count,
                      [user_check]FILE * stream);

        [fastcall] void test_fast_call([in]void* ptr);

        [stdcall] void test_std_call(void);

    };
};
```

Illegal Syntax:

```
enclave {

    untrusted {
        // Compiler warning without [cdecl,dllimport]
        size_t fwrite([in, size=size, count=count] const void* ptr,
                      size_t size,
                      size_t count,
                      [user_check] FILE * stream);

        // Compiler error without [stdcall]
        // Redefinition due to different type modifiers
        void test_std_call(void);
    };
};
```

Enclave Definition Language Libraries - Creating a Trusted Library with Import/Export Functions

Export and import functions can be implemented in external trusted libraries, akin to static libraries in the untrusted domain. The method of adding these functions to an enclave is by using the enclave definition language (EDL) library import mechanism.

Adding a library EDLfile to an enclave EDL file is done using the EDL keywords `from` and `import`.

The `from` keyword specifies the location of the library EDL file. Relative and full paths are accepted. Relative paths are relative to the location of the EDL file.

The `import` keyword specifies the functions to import. An asterisk (*) can be used to import all functions from the library. More than one function can be imported by writing a list of function names separated by commas.

Syntax

```
from “lib_filename.edl” import func_name, func2_name;
```

Or

```
from “lib_filename.edl” import *;
```

Example

```
enclave {
    from “secure_comms.edl” import send_email, send_sms;
    from "../../sys/other_secure_comms.edl" import *;
};
```

A library EDL file may import another EDL file, which in turn, may import another EDL file, creating a hierarchical structure as shown below:

```
// enclave.edl
enclave {
     from “other/file_L1.edl” import *;  // Import all functions
};

// Trusted library file_L1.edl
enclave {
     from "file_L2.edl" import *;

     trusted {
          public void test_int(int val);
     };
};

// Trusted library file_L2.edl
enclave {
     from "file_L3.edl" import *;

     trusted {
          public void test_ptr(int* ptr);
     };
};

// Trusted library file_L3.edl
enclave {

     trusted {
          public void test_float(float flt);
     };
};
```

Allowing Untrusted Functions to Call Trusted Functions

The default behavior is that an the untrusted functions (specified in the untrusted section in the EDL file) of an enclave can not call any of the trusted functions of this enclave.

If you want to grant an untrusted function access to an enclave exported function, specify this access using the `allow` keyword.

Syntax

```
untrusted {
    <function prototype> allow (func_name, func2_name, …);
};
```

Example

```
enclave {
    trusted {
        public void get_secret([out] secret_t* secret);
        void set_secret([in] secret_t* secret);
    };
    untrusted {
        void replace_secret(
            [in] secret_t* new_secret,
            [out] secret_t* old_secret)
            allow (set_secret);
    };
};
```

Public and Private ECALLs

Trusted functions are divided into public ECALLs and private ECALLs. Public ECALLs are those explicitly decorated with a `public` keyword, without this keyword, they will be treated as private ECALLs.

A public ECALL can always be directly called or called from a specific OCALL, whereas a private ECALL can only be called from a specific OCALL. Take the above EDL as an example, `set_secret` can only be called within the OCALL `replace_secret`.

An enclave EDL must have one or more public ECALLs, otherwise the Enclave functions cannot be called at all and `sgx_edger8r` will report an error in this case.

Enclave Configuration File

The enclave configuration file is an XML* based file containing the user defined parameters of an enclave. This XML file, as one part of the enclave project, contains the information of the enclave metadata. A tool named sgx_sign uses this file as an input to create the signature and metadata for the enclave. Here is an example of the configuration file:

```
<EnclaveConfiguration>
    <ProdID>100</ProdID>
    <ISVSVN>1</ISVSVN>
    <StackMaxSize>0x50000</StackMaxSize>
    <HeapMaxSize>0x100000</HeapMaxSize>
    <TCSNum>1</TCSNum>
    <TCSPolicy>1</TCSPolicy>
    <DisableDebug>0</DisableDebug>
    <MiscSelect>0</MiscSelect>
    <MiscMask>0xFFFFFFFF</MiscMask>
</EnclaveConfiguration>
```

The table below lists the elements defined in the configuration file. All of them are optional. Without a configuration file or if an element is not present in the configuration file, the default value will be used.

Table 13 Enclave Configuration Default Values

Tag	Description	Default Value
ProdID	ISV assigned Product ID.	0
ISVSVN	ISV assigned SVN.	0
TCSNum	The number of TCS. Must be greater than 0.	1
TCSPolicy	TCS management policy. 0 – TCS is bound to the untrusted thread. 1 – TCS is not bound to the untrusted thread.	1
StackMaxSize	The maximum stack size per thread. Must be 4KB aligned.	0x40000
HeapMaxSize	The maximum heap size for the process. Must be 4KB aligned.	0x100000
DisableDebug	Enclave cannot be debugged.	0 - Enclave can be debugged
MiscSelect	The desired Misc feature.	0
MiscMask	The mask bits for the Misc feature.	0xFFFFFFFF

The TCSNum must be greater than 0. StackMaxSize and HeapMaxSize must be 4K byte aligned. MiscSelect and MiscMask are for future functional extension. Currently, MiscSelect must be 0. Otherwise the corresponding enclave may not be loaded successfully.

To avoid wasting the valuable protected memory resource, you can properly adjust the StackMaxSize and HeapMaxSize by using the measurement tool sgx_emmt. See Enclave Memory Measurement Tool for details.

A Visual Studio Add-in named **SGX Configuration** is provided for users to edit their configuration file conveniently. See Using SGX Configuration Add-in for details.

Enclave Project Configurations

Depending on the stage an enclave developer is at, he must choose one of the following project configurations to build an enclave:

- Simulation: The simulation mode works in the same way as the debug mode except the fact that true hardware is not exercised, instead the Intel® SGX instructions are simulated in software. Single-step signing is the default method to sign a simulation enclave.
- Debug: When the **Debug** configuration option is selected for an enclave project in Microsoft* Visual Studio, the enclave is compiled in the *debug* mode and the resulting enclave file will contain debug information and symbols. Choosing this project configuration also allows the enclave to be launched in the *enclave debug* mode. This is facilitated by enabling the SGX_DEBUG_FLAG that is passed as one of the parameters to the sgx_create_enclave function. Single-step

method is the default signing method for this project configuration. The signing key used in this mode does not need to be white-listed.

- Prerelease: When you choose the **Prerelease** configuration option for an enclave project, Visual Studio will build the enclave in *release* mode with compiler optimizations applied. Under this configuration, the enclave is launched in *enclave debug* mode. A preprocessor flag `EDEBUG` is defined in the preprocessor settings of the Microsoft Visual Studio enclave project for this mode. When the `EDEBUG` preprocessor flag is defined, it enables the `SGX_DEBUG_FLAG`, which in turn, launches the enclave in the *enclave debug* mode. Single-step method is also the default signing method for the Prerelease project configuration. Like in the Debug configuration, the signing key does not need to be white-listed either.
- Release: The **Release** configuration option for a Visual Studio enclave project compiles the enclave in the *release* mode and launches the enclave in the *enclave release* mode. This is done by disabling the `SGX_DEBUG_FLAG`. `SGX_DEBUG_FLAG` is only enabled when `NDEBUG` is not defined or `EDEBUG` is defined. In the debug configuration `NDEBUG` is undefined and hence `SGX_DEBUG_FLAG` is enabled. In the prerelease configuration `NDEBUG` and `EDEBUG` are both defined, which enables `SGX_DEBUG_FLAG`. In the release mode, configuration `NDEBUG` is defined and hence it disables `SGX_DEBUG_FLAG` thereby launching the enclave in *enclave release* mode. Two-step method is the default signing method for the Release configuration. The enclave needs to be signed with a white-listed key.

For additional information on the different enclave signing methods, see The Enclave Signing Tool and Enclave Signing Examples

Load and Unload an Enclave

Enclave source code is built as a dynamic link library. To use an enclave, the enclave.dll should be loaded into enclave memory by calling the API `sgx_create_enclave()`. The enclave.dllmust be signed by sgx_sign.exe. When loading an enclave for the first time, the loader will get a launch token and save it back to the in/out parameter `token`. The user can save the launch token into a file, so that when loading an enclave for the second time, the application can get the launch token from the saved file. Providing a valid launch token can enhance the load performance. To unload an enclave, the user must call `sgx_destroy_enclave()` interface with parameter `sgx_enclave_id_t`.

The sample code to load and unload an Enclave is shown below.

```
#include <stdio.h>
#include <tchar.h>
#include "sgx_urts.h"

#define ENCLAVE_FILE _T("Enclave.signed.dll")

int main(int argc, char* argv[])
{
     sgx_enclave_id_t   eid;
     sgx_status_t       ret   = SGX_SUCCESS;
     sgx_launch_token_t token = {0};
     int updated = 0;

     // Create the Enclave with above launch token.
     ret = sgx_create_enclave(ENCLAVE_FILE, SGX_DEBUG_FLAG, &token, &up-
     dated, &eid, NULL);
     if (ret != SGX_SUCCESS) {
```

```
        printf("App: error %#x, failed to create enclave.\n", ret);
        return -1;
    }

    // A bunch of Enclave calls (ECALL) will happen here.


    // Destroy the enclave when all Enclave calls finished.
    if(SGX_SUCCESS != sgx_destroy_enclave(eid))
        return -1;

    return 0;
}
```

Handling Power Events

The protected memory encryption keys that are stored within an SGX-enabled CPU are destroyed with every power event, including suspend and hibernation.

Thus, when a power transition occurs, the enclave memory will be removed and all enclave data will not be accessible after that. As a result, when the system resumes, any subsequent ECALL will fail returning the error code `SGX_ERROR_ENCLAVE_LOST`. This specific error code indicates the enclave is lost due to a power transition.

An SGX application should have the capability to handle any power transition that might occur while the enclave is loaded in protected memory. To handle the power event and resume enclave execution with minimum impact, the application must be prepared to receive the error code `SGX_ERROR_ENCLAVE_LOST` when an ECALL fails. When this happens, one and only one thread from the application must destroy the enclave, `sgx_destroy_enclave()`, and reload it again, `sgx_create_enclave()`. In addition, to resume execution from where it was when the enclave was destroyed, the application should periodically seal and save enclave state information on the platform and use this information to restore the enclave to its original state after the enclave is reloaded.

The Power Transition sample code included in the SDK demonstrates this procedure.

NOTE:

On Windows* 10, an SGX application must call `sgx_destroy_enclave()` for the OS to reclaim protected memory or EPC pages from enclaves that have been removed due to power events. Not destroying an enclave will result in EPC memory leakage that could prevent subsequent enclaves from loading. When this happens `sgx_create_enclave()` will return the error code `SGX_ERROR_OUT_OF_EPC`.

Loading Untrusted SGX DLLs

The SGX DLLs shipped with the PSW (`sgx_urts.dll`and `sgx_uae_service.dll`) are installed in the system directory. You must lock down the SGX application installation directory. Otherwise, you must explicitly load these two DLLs.

Suppose an attacker gains control over the directory where the application is installed and inserts a malicious copy of an SGX DLL in that directory. If the application implicitly loads the SGX DLLs, then the bad copy will get loaded before the original SGX DLLs from the system path.

To make sure that an SGX application is loading the SGX DLLs from the system directory, the application should explicitly load the two DLLs in the following order:

1. sgx_uae_service.dll
2. sgx_urts.dll

Intel® Software Guard Extensions Sample Code

After installing the Intel® Software Guard Extensions Evaluation SDK, the sample code is under the sub-folder *src*.

You can open the sample projects in Microsoft* Visual Studio* 2012. It is suggested to use Intel® C++ Compiler XE 13.0 to compile the sample projects, which is the default setting in the project properties.

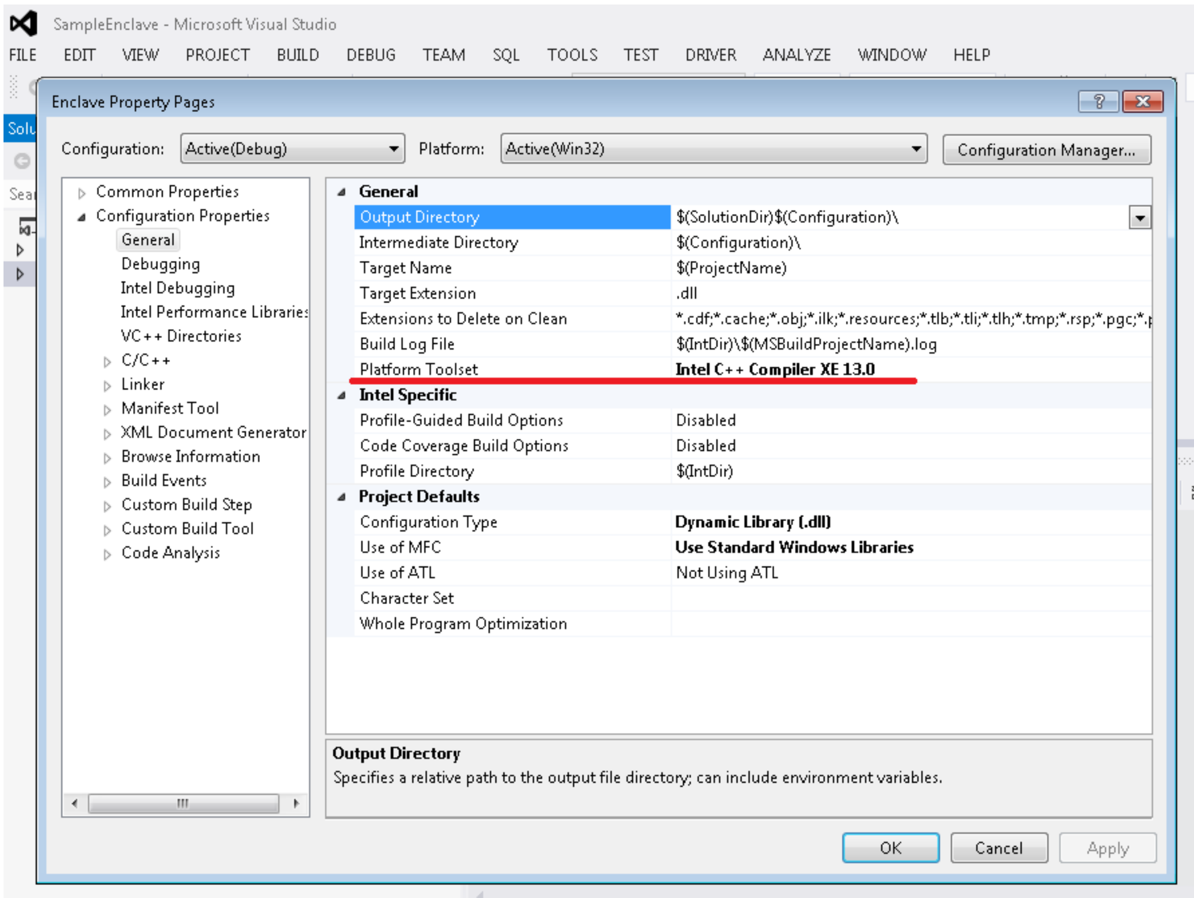

Figure 15 Using Intel® C++ Compiler XE 13.0 for Sample Projects

- The *SampleEnclave* project shows how to create an enclave.
- The *PowerTransition* project shows how to handle the power transition for the Intel® SGX project.
- The *LocalAttestation* project shows how to use the Intel Elliptical Curve Diffie-Hellman key exchange library to establish a trusted channel between two enclaves running on the same platform.
- The *RemoteAttestation* project shows how to use the Intel remote attestation and key exchange library in the remote attestation process.
- The *SealedData* project demonstrates how to use the APIs to encrypt and integrity-protect enclave secrets to store them on disk.

- The *X509* project shows how Intel® SGX can be used along with OpenSSL* to verify an X509 certificate safely.

Sample Enclave

The project *SampleEnclave* is designed to show you how to write an enclave from scratch. This topic demonstrates the following basic aspects of enclave features:

- Initialize and destroy an enclave
- Create ECALLs and/or OCALLs
- Call trusted libraries inside the enclave

The code is shipped with the Intel® Software Guard Extensions Evaluation SDK in `$(SGXSDKInstallPath)src\SampleEnclave`. You can open the project through Microsoft* Visual Studio 2012.

NOTE:

If the sample project is located in a system directory, administrator privilege is required to open it. You can copy the project folder to your directory if administrator permission cannot be granted.

Configure and Enable Intel® SGX

Some OEM systems support configuration and enabling of Intel® SGX in the BIOS via an SW Control Interface. The Intel SGX PSW exposes an API that ALL applications should call prior to creating an application. The API `sgx_enable_device` configures and enables the Intel SGX device if the platform has NOT been previously enabled. If the BIOS configures Intel SGX as result of the call, then a reboot is required for the BIOS configuration to take affect (Intel SGX will not be available for use until after the reboot). Please, refer to the `query_sgx_status` function in the Sample Application for use of this API. For additional details on `sgx_enable_device`, refer to the Library Functions and Type Reference section of this document.

Initialize an Enclave

Before establishing any trusted transaction between an application and an enclave, the enclave itself needs to be correctly created and initialized. The procedure is demonstrated as shown in this section.

Retrieve the Saved Token

If the launch token was saved in a previous transaction, it can be retrieved and used for subsequent enclave initializations. The launch token should be saved in a per-user directory or a registry entry in case it would be used in a multi-user environment.

For example, the token can be saved in either of the following locations:

- `CSIDL_LOCAL_APPDATA` - the file system directory where application-specific data is stored
- `HKEY_CURRENT_USER` - the registry entry that contains the profile for the user who is currently logged on to the computer.

See http://msdn.microsoft.com/en-us/library/windows/desktop/bb762494(v=vs.85).aspx for details about `CSIDL_LOCAL_APPDAT`.

Create an Enclave

After the launch token is retrieved, developers are able to create an enclave instance by calling sgx_create_enclave provided by the uRTS library. Any error returned by this function should be handled. For example, you can simply convert them to meaningful error messages. See sgx_create_enclave for details. Particularly, you need to handle power transitions during enclave initialization, which is demonstrated in the Power Transition.

Store the Updated Token

After the enclave is correctly created and initialized, you may need to save the token if it has been updated. The fourth parameter of sgx_create_enclave indicates whether or not an update has been performed.

ECALL/OCALL Functions

The ECALL is an entry point for an application to utilize Intel® SGX capabilities; it not only includes a functional declaration in the trusted section of an EDL file, but also an actual functional implementation inside the enclave.

An OCALL provides an access point that enables you to use operating system capabilities outside the enclave such as system calls, I/O operations, and so on. A public ECALL is mandatory for writing an enclave, while OCALLs are optional.

This sample demonstrates basic EDL syntax used by ECALL/OCALL functions, as well as using trusted libraries inside the enclave. You may see Enclave Definition Language Syntax for syntax details and Trusted Libraries for C/C++ support.

EDL Syntax

Syntax Category	Attributes Covered
Array	[], isary
Data Types	struct, enum, union, char, int, float, double, size_t, wchar_t
Function	public, private, cdecl, dllimport, allow
Pointer	user_check, in, out, string, const, size, count, isptr, readonly, sizefunc

Trusted Libraries

Library Category	Functionalities Covered
Standard C Library	Memory Allocation and Free
Standard C++ Library	C++ Exception, STL <map> Template
Trusted Thread Library	Mutex, Condition Variable

Destroy an Enclave

To release the enclave memory, you need to invoke sgx_destroy_enclave provided by the uRTS library. It will recycle the EPC memory and untrusted resources used by that enclave instance.

Power Transition

If a power transition occurs, the enclave memory will be removed and all the enclave data will be inaccessible. Consequently, when the system is resumed, each of the in-process ECALLS and the subsequent ECALLs will fail with the error code SGX_ERROR_ENCLAVE_LOST which indicates the enclave is lost due to a power transition.

An Intel® Software Guard Extensions project should have the capability to handle the power transition which might impact its behavior. The project named *PowerTransition* describes one method of developing Intel® Software Guard Extensions projects that handle power transitions. See ECALL-Error-Code Based Retry for more info.

PowerTransition demonstrates the following scenario: an enclave instance is created and initialized by one main thread and shared with three other child threads; The three child threads repeatedly ECALL into the enclave, manipulate secret data within the enclave and backup the corresponding encrypted data outside the enclave; After all the child threads finish, the main thread destroys the enclave and frees the associated system resources. If a power transition happens, one and only one thread will reload the enclave and restore the secret data inside the enclave with the encrypted data that was saved outside and then continues the execution.

The *PowerTransition* sample code is shipped with the Intel® Software Guard Extensions Evaluation SDK. You can find the source code in the `$(SGXSDKInstallPath) src\PowerTransition` directory. The sample code can be built with Microsoft* Visual Studio 2012 using the corresponding project in Microsoft* Visual Studio 2012.

NOTE:

If the sample project locates in a system directory, administrator privilege is required to open it. You can copy the project folder to your directory if administrator permission cannot be granted.

ECALL-Error-Code Based Retry

After a power transition, an Intel® SGX error code SGX_ERROR_ENCLAVE_LOST will be returned for the current ECALL. To handle the power transition and continue the project without impact, you need to destroy the invalid enclave to free resources first and then retry with a newly created and initialized enclave instance, as depicted in the following figure.

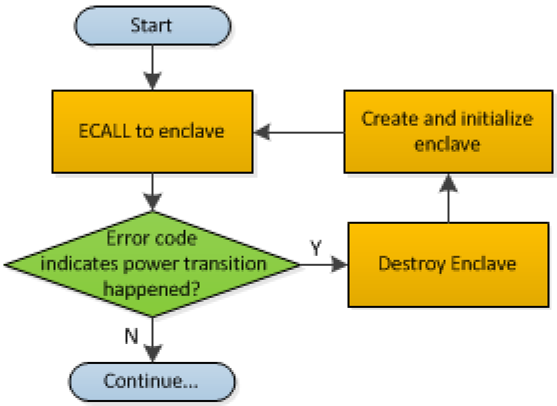

Figure 16 Power Transition Handling Flow Chart

ECALLs in Demonstration

PowerTransition demonstrates handling the power transition in two types of ECALLs:

1. Initialization ECALL after enclave creation.
2. Normal ECALL to manipulate secrets within the enclave.

Initialization ECALL after Enclave Creation

PowerTransition illustrates one initialization ECALL after enclave creation which is shown in the following figure:

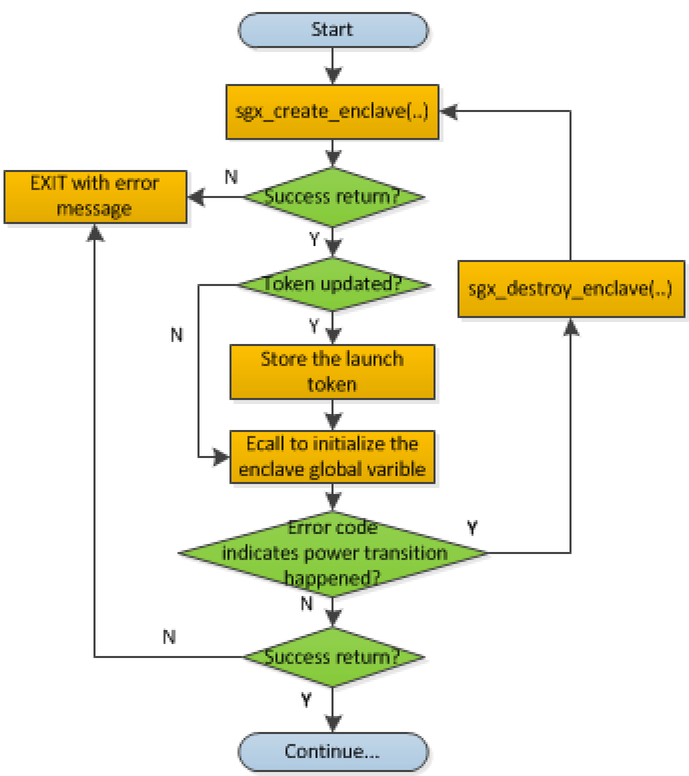

Figure 17 Enclave Initialization ECall after Enclave Creation Flow Chart

sgx_create_enclave is a key API provided by the uRTS library for enclave creation. For sgx_create_enclave, a mechanism of power transition handling is already implemented in the uRTS library. Therefore, it is unnecessary to manually handle power transition for this API.

NOTE:

To concentrate on handling a power transition, PowerTransition assumes the enclave file and the launch token are located in the same directory as the application. See Sample Enclave for how to store the launch token properly.

Normal ECALL to Process Secrets within the Enclave

This is the most common ECALL type into an enclave. *PowerTransition* demonstrates the power transition handling for this type of ECALL in a child thread after the enclave creation and initialization by the main thread, as depicted in the figure below. Since the enclave instance is shared by the child threads, it is required to make sure one and only one child thread to re-creates and re-initializes the enclave instance after the power transition and the others utilize the re-created enclave instance directly. *PowerTransition* confirms this point by checking whether the Enclave ID is updated.

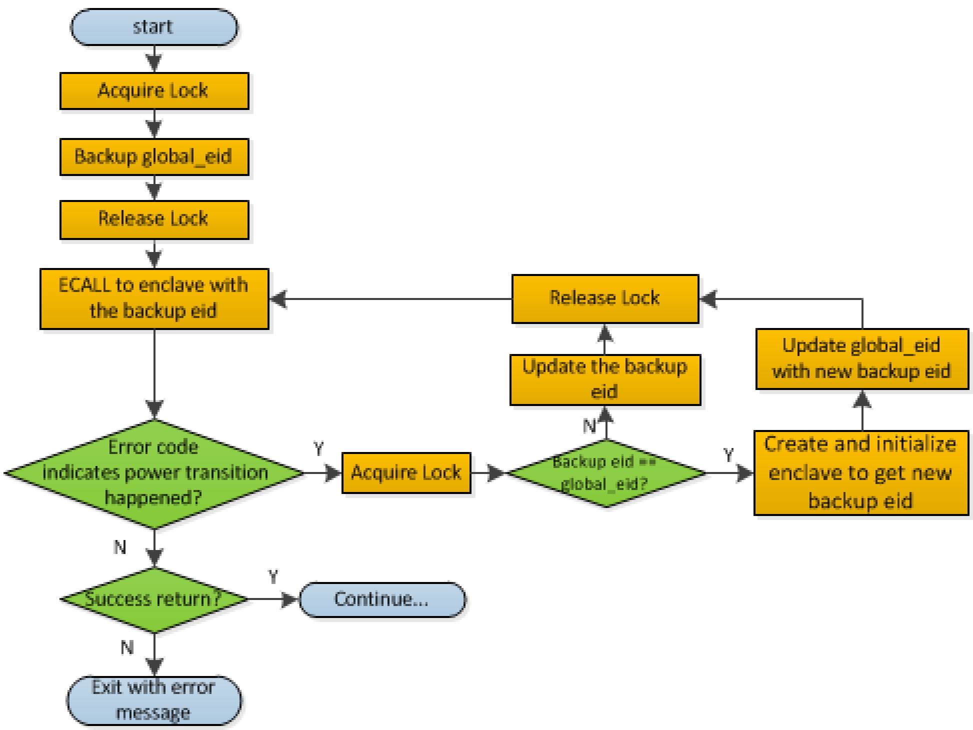

Figure 18 Regular ECALL Flow Chart

NOTE:

During the ECALL process, it is recommended to back up the confidential data as cipher text outside the enclave frequently. Then we can use the backup data to restore the enclave to reduce the power transition impacts.

Attestation

In the Intel® Software Guard Extensions architecture, attestation refers to the process of demonstrating that a specific enclave was established on the platform. The Intel® SGX Architecture provides two attestation mechanisms:

- One creates an authenticated assertion between two enclaves running on the same platform referred to as local attestation.
- The second mechanism extends local attestation to provide assertions to 3rd parties outside the platform referred to as remote attestation. The remote attestation process leverages a quoting service.

The Intel® Software Guard Extensions Evaluation SDK provides APIs used by applications to implement the attestation process.

Local Attestation

Local attestation refers to two enclaves on the same platform authenticating to each other using the SGX REPORT mechanism before exchanging information. In an Intel® SGX application, multiple enclaves might collaborate to perform certain functions. After the two enclaves verify the counterpart is trustworthy, they can exchange information on a protected channel, which typically provides confidentiality, integrity and replay protection. The local attestation and protected channel establishment uses the REPORT based Diffie-Hellman Key Exchange* protocol.

You can find a sample solution shipped with the Intel® Software Guard Extensions Evaluation SDK at `$(SGXSDKInstallPath)src\LocalAttestation` directory. To compile, you only need to open the project with Microsoft* Visual Studio 2012.

NOTE:

If the sample project locates in a system directory, administrator privilege is required to open it. You can copy the project folder to your directory if administrator permission cannot be granted.

The sample code shows an example implementation of local attestation, including protected channel establishment and secret message exchange using enclave to enclave function call as an example.

Diffie-Hellman Key Exchange Library and Local Attestation Flow

The local attestation sample in the SDK uses the Diffie-Hellman (DH) key exchange library to establish a protected channel between two enclaves. The DH key exchange APIs are described in `sgx_dh.h`. The key exchange library is part of the Intel® SGX application SDK trusted libraries. It is statically linked with the enclave code and exposes APIs for the enclave code to generate and process local key exchange protocol messages. The library is combined with other libraries and is built into the final library called sgx_tservice.lib that is part of the SDK release.

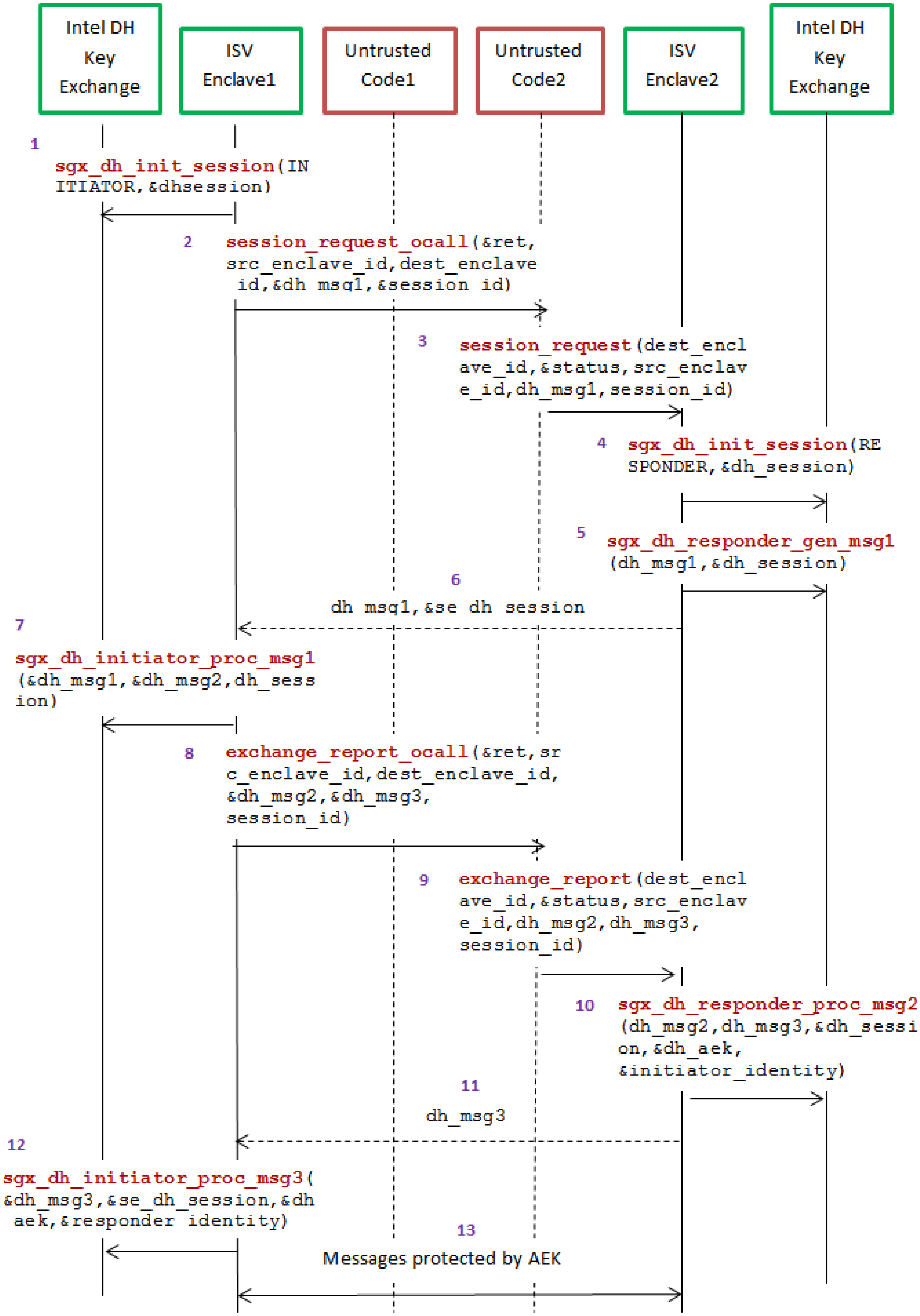
Intel DH Key Exchange
ISV Enclave1
Untrusted Code1
Untrusted Code2
ISV Enclave2
Intel DH Key Exchange
1
sgx_dh_init_session(INITIATOR, &dhsession)
2
session_request_ocall(&ret, src_enclave_id, dest_enclave id, &dh msg1, &session id)
3
session_request(dest_enclave_id, &status, src_enclave_id, dh_msg1, session_id)
4
sgx_dh_init_session(RESPONDER, &dh_session)
5
sgx_dh_responder_gen_msg1
(dh_msg1, &dh_session)
6
dh msg1, &se dh session
7
sgx_dh_initiator_proc_msg1
(&dh_msg1, &dh_msg2, dh_session)
8
exchange_report_ocall(&ret, src_enclave_id, dest_enclave_id, &dh_msg2, &dh_msg3, session_id)
9
exchange_report(dest_enclave_id, &status, src_enclave_id, dh_msg2, dh_msg3, session_id)
10
sgx_dh_responder_proc_msg2
(dh_msg2, dh_msg3, &dh_session, &dh_aek, &initiator_identity)
11
dh_msg3
12
sgx_dh_initiator_proc_msg3(&dh_msg3, &se_dh_session, &dh aek, &responder identity)
13
Messages protected by AEK

Figure 19 Local Attestation Flow with the DH Key Exchange Library

The figure above represents the usage of DH key exchange library. A local attestation flow consists of the following steps:

1. ISV Enclave 1 calls the Intel ECDH key exchange library to initiate the session with the initiator role.
2. The Enclave 1 does an OCALL into the untrusted code requesting the Diffie-Hellman Message 1 and session id.
3. The untrusted code does an ECALL into Enclave 2.
4. Enclave 2 in turn calls the ECDH key exchange library to initiate the session with the responder role.
5. Enclave 2 calls the key exchange library to generate DH Message 1 `ga || TARGETINFO` Enclave 2.
6. DH Message 1 is sent back from Enclave 2 to Enclave 1 through an ECALL return to the untrusted code followed by an OCALL return into Enclave 1.
7. Enclave 1 processes the Message 1 using the key exchange library API and generates DH Message 2 `gb||[Report Enclave 1(h(ga || gb))]SMK`.
8. DH Message 2 is sent to the untrusted side through an OCALL.
9. The untrusted code does an ECALL into Enclave 2 giving it the DH Message 2 and requesting DH Message 3.
10. Enclave 2 calls the key exchange library API to process DH Message 2 and generates DH Message 3 `[ReportEnclave2(h(gb || ga)) || Optional Payload]SMK`.
11. DH Message 3 is sent back from Enclave2 to Enclave1 through an ECALL return to the untrusted code followed by an OCALL return into Enclave 1.
12. Enclave 2 uses the key exchange library to process DH Message 3 and establish the session.
13. Messages exchanged between the enclaves are protected by the AEK.

Protected Channel Establishment

The following figure illustrates the interaction between two enclaves, namely the source enclave and the destination enclave, to establish a session. The application initiates a session between the source enclave and the destination enclave by doing an ECALL into the source enclave, passing in the enclave id of the destination enclave. Upon receiving the enclave id of the destination enclave, the source enclave does an OCALL into the core untrusted code which then does an ECALL into the destination enclave to exchange the messages required to establish a session using ECDH Key Exchange* protocol.

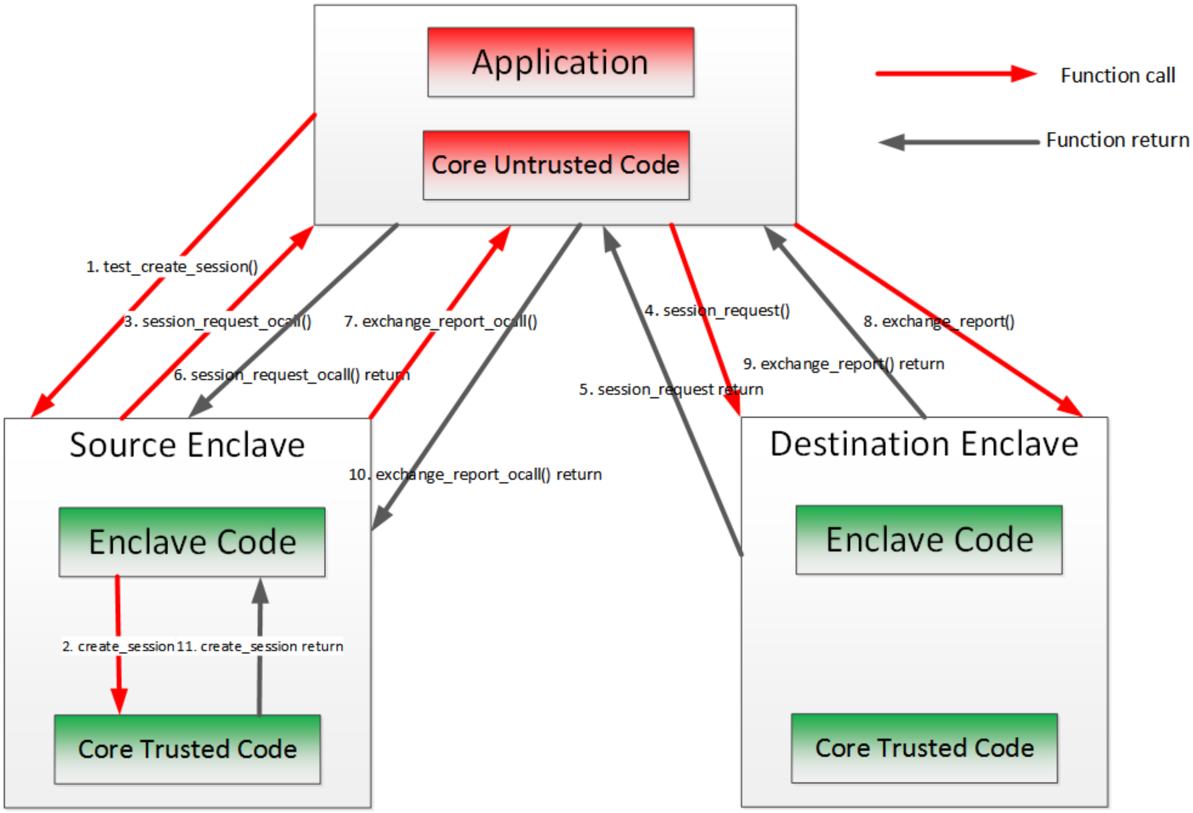

Figure 20 Secure Channel Establishment Flow with the DH Key Exchange Library

Secret Message Exchange and Enclave to Enclave Call

The following figure illustrates the message exchange between two enclaves. After the establishment of the protected channel, session keys are used to encrypt the payload in the message(s) being exchanged between the source and destination enclaves. The sample code implements interfaces to encrypt the payload of the message. The sample code also shows the implementation of an enclave calling a function from another enclave. Call type, target function ID, total input parameter length and input parameters are encapsulated in the payload of the secret message sent from the caller (source) Enclave and the callee (destination) enclave. As one enclave cannot access memory of another enclave, all input and output parameters, including data indirectly referenced by a parameter needs to be marshaled across the two enclaves. The sample code uses Intel® SGX Evaluation SDK trusted cryptographic library to encrypt the payload of the message. Through such encryption, message exchange is just the secret and in case of the enclave to enclave call is the marshaled destination enclave's function id, total parameter length and all the parameters. The destination enclave decrypts the payload and calls the appropriate function. The results of the function call are encrypted using the session keys and sent back to the source enclave.

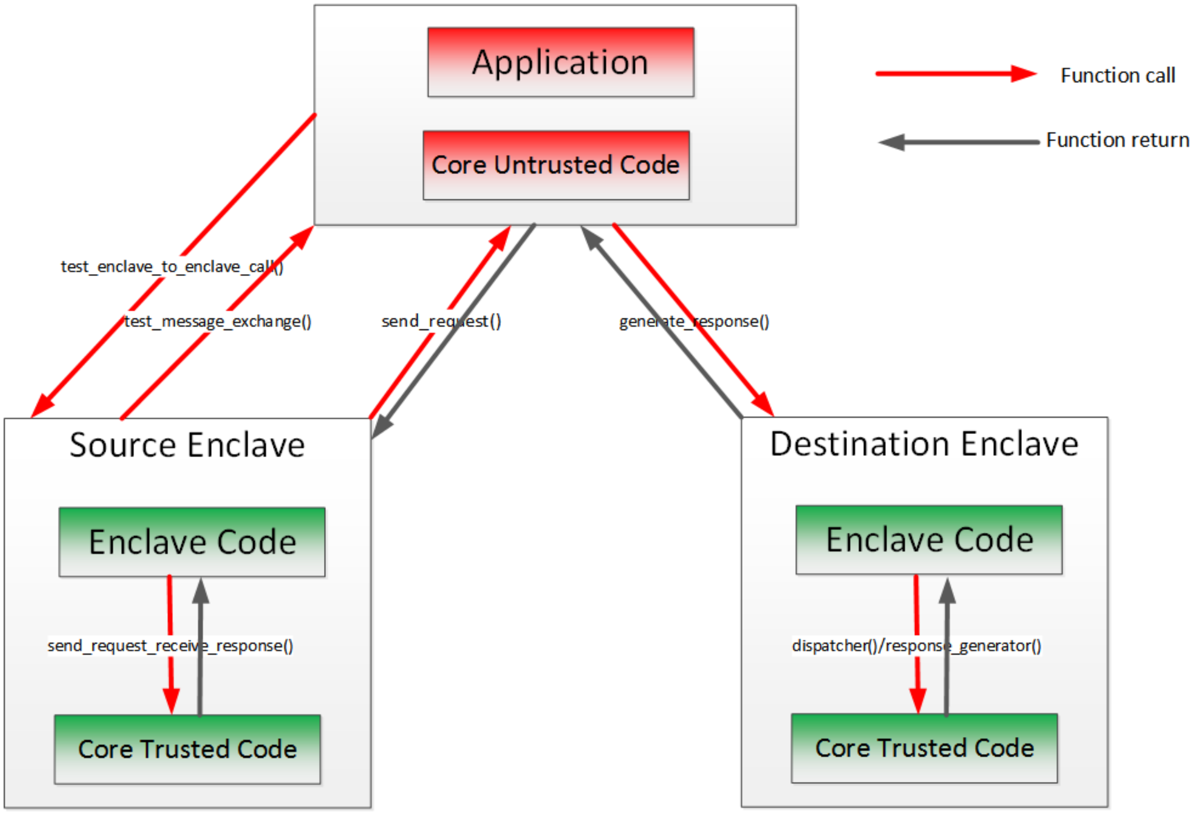

Figure 21 Secret Message Exchange Flow with the DH Key Exchange Library

Remote Attestation

Generally speaking, Remote Attestation is the concept of a HW entity or of a combination of HW and SW gaining the trust of a remote provider or producer of some sort. With Intel® SGX, Remote Attestation software includes the app's enclave and the Intel-provided Quoting Enclave (QE) and Provisioning Enclave (PvE). The attestation HW is the Intel® SGX enabled CPU.

Remote Attestation alone is not enough for the remote party to be able to securely deliver their service (secrets or assets). Securely delivering services also requires a secure communication session. Remote Attestation is used during the establishment of such a session. This is analogous to how the familiar SSL handshake includes both authentication and session establishment.

The Intel® Software Guard Extensions Evaluation SDK includes sample code showing:

- How an application enclave can attest to a remote party.
- How an application enclave and the remote party can establish a secure session.

The SDK includes a remote session establishment or key exchange (KE) libraries that can be used to greatly simplify these processes.

You can find the sample code for remote attestation in the directory `$(SGXSDKInstallPath) src\RemoteAttestation`.

NOTE:

The Intel® Attestation Service has been activated. A sandbox version of Intel Attestation Service is supported to enable development in an ISV's application server for Intel® SGX attestation. Refer to the Intel® Attestation Service documentation for information on how to establish the communication between the ISV Application Server and Intel Attestation Server.

NOTE:

If the sample project is located in a system directory, administrator privilege is required to open it. You can copy the project folder to your directory if administrator permission cannot be granted.

Intel® SGX uses an anonymous signature scheme, Enhanced Privacy ID (EPID), for authentication (for example, attestation). The supplied key exchange libraries implement a Sigma-like protocol for session establishment. Sigma is a protocol that includes a Diffie-Hellman key exchange, but also addresses the weaknesses of DH. The protocol Intel® SGX uses differs from the Sigma protocol that's used in IKE v1 and v2 in that the Intel® SGX platform uses EPID to authenticate while the service provider uses PKI. (In Sigma, both parties use PKI.) Finally, the KE libraries require the service provider to use an ECDSA, not an RSA, key pair in the authentication portion of the protocol and the libraries use ECDH for the actual key exchange.

Remote Key Exchange (KE) Libraries

The RemoteAttestation sample in the SDK uses the remote KE libraries as described above to create a remote attestation of an enclave, and uses that attestation during establishment of a secure session (a key exchange).

There are both untrusted and trusted KE libraries. The untrusted KE library is provided as a static library, `sgx_ukey_exchange[mt].lib`. The Intel® SGX application needs to link with this library and include the header file `sgx_ukey_exchange.h`, containing the prototypes for the APIs that the KE trusted library exposes.

The trusted KE library is also provided as a static library. As a trusted library, the process for using it is slightly different than that for the untrusted KE library. The main difference relates to the fact that the trusted KE library exposes ECALLs called by the untrusted KE library. This means that the library has a corresponding EDL file, `sgx_tkey_exchange.edl`, which has to be imported in the EDL file for the application enclave that uses the library. We can see this in code snippet below, showing the complete contents of `app_enclave.edl`, the EDL file for the app enclave in the sample code.

```
enclave {
    from "sgx_tkey_exchange.edl" import *;
    include "sgx_key_exchange.h"
    include "sgx_trts.h"
    trusted {
        public sgx_status_t enclave_init_ra(
                    int b_pse,
                    [out] sgx_ra_context_t *p_context);
        public sgx_status_t enclave_ra_close(
                    sgx_ra_context_t context);
    };
};
```

It's worth noting that `sgx_key_exchange.h` contains types specific to remote key exchange and must be included as shown above as well as in the untrusted code of the application that uses the enclave. Finally, `sgx_tkey_exchange.h` is a header file that includes prototypes for the APIs that the trusted library exposes, but that are not ECALLs, i.e., APIs called by ISV code in the application enclave.

Remote Attestation and Protected Session Establishment

This topic describes the functionality of the remote attestation sample in detail.

NOTE:

In the sample code, the service provider is modeled as a DLL, `service_provider.dll`. The sample service provider does not depend on Intel® SGX headers, type definitions, libraries, and so on. This was done to demonstrate that the Intel SGX is not required in any way when building a remote attestation service provider.

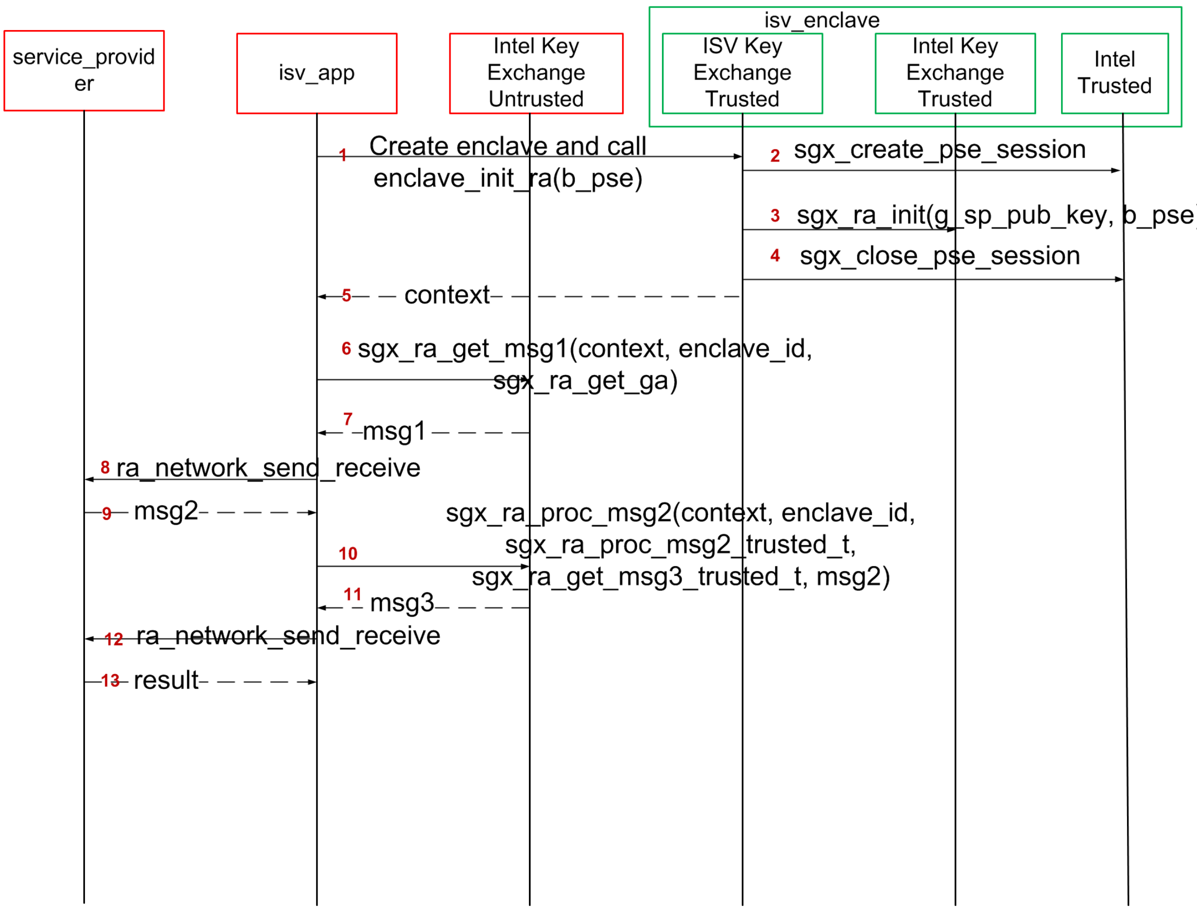

Figure 22 Remote Attestation and Trust Channel Establishment Flow

An Intel® Software Guard Extensions (Intel® SGX) application would typically begin by requesting service (for example, media streaming) from a service provider (SP) and the SP would respond with a challenge. This is not shown in the figure. The figure begins with the app's reaction to the challenge.

1. The flow starts with the app entering the enclave that will be the endpoint of the KE, passing in `b_pse`, a flag indicating whether the app/enclave uses Platform Services.
2. If b_pse is true, then the isv enclave shall call trusted AE support library with `sgx_create_pse_session()` to establish a session with PSE.
3. Code in the enclave calls `sgx_ra_init()`, passing in the SP's ECDSA public key, `g_sp_pub_key`, and `b_pse`. The integrity of `g_sp_pub_key` is a public key is important so this value should just be built into isv_enclave.
4. Close PSE session by `sgx_close_pse_session()` if a session is established before. The requirement is that, if the app enclave uses Platform Services, the session with the PSE must already be established before the app enclave calls `sgx_ra_init()`.
5. `sgx_ra_init()` returns the KE context to the app enclave and the app enclave returns the context to the app.
6. The app calls `sgx_ra_get_msg1()`, passing in this KE's context. Figure 3 shows the app also passing in a pointer to the untrusted proxy corresponding to `sgx_ra_get_ga`, exposed by the TKE. This reflects the fact that the names of untrusted proxies are enclave-specific.
7. `sgx_ra_get_msg1()` builds an S1 message = (ga || GID) and returns it to the app.
8. The app sends S1 to the service provider (SP) by `ra_network_send_receive()`, it will call `sp_ra_proc_msg1_req()` to process S1 and generate S2.
9. Application eventually receives `S2 = gb || SPID || TYPE || SigSP(gb, ga) || CMACSMK(gb || SPID || TYPE || SigSP(gb, ga)) || SigRL`.
10. The application calls `sgx_ra_proc_msg2()`, passing in S2 and the context.
11. The code in `sgx_ra_proc_msg2()` builds`S3 = CMAC(SMKCMAC,M)||M` where `M = ga ||PS_SECURITY_PROPERTY|| QUOTE` and returns it. Platform Services Security Information is included only if the app/enclave uses Platform Services.
12. Application sends the msg3 to the SP by `ra_network_send_receive()`, and the SP verifies the msg3.
13. SP returns the verification result to the application.

At this point, a session has been established and keys exchanged. Whether the service provider thinks the session is secure and uses it depends on the security properties of the platform as indicated by the S3 message. If the platform's security properties meet the service provider's criteria, then the service provider can use the session keys to securely deliver a secret and the app enclave can consume the secret any time after it retrieves the session keys by calling `sgx_ra_get_keys()` on the trusted KE library. This is not shown in the figure, nor is the closing of the session. Closing the session requires entering the app enclave and calling `sgx_ra_close()` on the trusted KE library, among other app enclave-specific cleanup.

Debugging a Remote Attestation Service Provider

As an ISV writing the remote attestation service provider, you may want to debug the message flow. One way to do this would be to provide pre-generated messages that can be replayed and verified. However, not that `S1 message = (GID || ga)` includes the random component `ga` generated inside an enclave. Also, the remote attestation service provider generates a random public+private key pair as part of its `msg2` generation, but without any interaction with Intel® SGX. Finally, each of these has state or context that is associated with cryptographic operations and is used to ensure that certain calls being made are in the correct order and that the state is consistent. These characteristics help protect the remote attestation flow against attacks, but also make it more difficult to replay pre-generated messages.

To overcome these, the cryptographic library is modified and used (only) by the sample service provider. Any time that key generation, signing, or other operation requests a random number, the number `9` is returned. This means that the crypto functions from `sample_libcrypto.lib` are predictable and cryptographically weak. If we can replay `msg1` send from the `isv_app`, the sample `service_provider.dll` will always generate the exact same `msg2`. We now have a sufficient system to replay messages sent by the `isv_app` and have it verify that the responses sent by the remote service are the expected ones.

To replay messages and exercise this verification flow, pass in `1` or `2` as a command-line argument when running the sample application `isv_app`. The `isv_app` will ignore errors generated by the built-in checks in the Intel SGX. Developers wishing to debug their remote attestation service provider should be able to temporarily modify their cryptographic subsystem to behave in a similar manner as the `sample_libcrypto.lib` and replay the pre-computed messages stored in `sample_messages.h`. The responses from their own remote attestation service provider should match the ones generated by ours, which are also stored in `sample_messages.h`.

NOTE

Do not use the sample cryptographic library provided in this sample in production code.

Sealed Data

The Intel® SGX SDK provides APIs to encrypt and integrity-protect enclave secrets to store them outside the enclave, such as on disk. The Intel® SGX Platform SW provides Monotonic Counter and Trusted Time service to ISV enclaves. The Monotonic Counter can be used to implement replay-protected policy, and the Trusted Time can be used to enforce time based policy. Both of them are in a form of Sealed Data. The requirement of replay-protected data blob and time based policy data blob is quite subtle. The Intel® SGX SDK will provide reference code to help ISV to implement them correctly.

The sample code *SealedData* is shipped with the Intel® Software Guard Extensions Evaluation SDK in `$(SGXSDKInstallPath)src\SealedData` folder.To compile, you only need to open the project with Microsoft* Visual Studio 2012.

NOTE:

If the sample project is located in a system directory, administrator privilege is required to modify it. You can copy the project folder to your directory if administrator permission cannot be granted.

Replay Protected Policy

In Enterprise Rights Management (ERM) type usages, an offline activity log might need to be maintained and periodically audited by the enterprise, for example, depending on whether and/or how many times a secret document is viewed or printed offline. If the offline activity log is tampered with or deleted, the ERM application will disable the offline use capability. A functional secure document viewing ERM application is quite complex, involving credential verification, document key provisioning, secure document rendering, secure display and many other security processes.

The Replay Protected policy sample code will not implement a full secure document viewing functionality, instead, it will demonstrate:

- Initializing a replay protected policy, to create an offline activity log together with a secret, protected by a Monotonic Counter;
- Verifying and updating the replay protected policy, to verify and update the activity log before the secret can be used to perform a function.

- Deleting the replay protected policy, to delete the activity log and the associated Monotonic Counter after the secret is invalidated.

Initializing a Policy

1. The Enclave creates a new Monotonic Counter using `sgx_create_monotonic_counter`.
2. The Enclave fills the activity log with the sample usage secret and usage data, the Monotonic `Counter_UUID` and the Monotonic `Counter_Value` returned by `sgx_create_monotonic_counter`.
3. The Enclave seals the activity log into sealed data using `sgx_seal_data`.

Verifying a Policy

1. The Enclave verifies and decrypts the sealed data using `sgx_unseal_data`
2. The Enclave retrieves the current Monotonic Counter value of the associated Monotonic Counter using `sgx_read_monotonic_counter`. If it fails, abort the operation.
3. The Enclave verifies the Monotonic `Counter_Value` returned by `sgx_read_monotonic_counter` is the same as the Monotonic `Counter_Value` in the activity log.
4. The Enclave releases the secret to perform functions.

Updating a Policy

1. The Enclave verifies activity log.
2. The Enclave checks that the secret and usage data inside the activity log has not been invalidated or expired, for example, by comparing the use count in the Activity Log against a predetermined threshold. If the secret is invalidated or expired, the function that requires the secret will not be rendered.
3. The Enclave Increases the Monotonic Counter value of the associated MC using `sgx_increment_monotonic_counter`. If it fails, abort the operation.
4. The Enclave verifies the Monotonic Counter value returned in `sgx_increment_monotonic_counter` is equal to the old value, previously returned by `sgx_read_monotonic_counter`, plus one.
5. The Enclave updates the activity log and the Monotonic `Counter_Value`.
6. The Enclave seals the activity log into Sealed Data using `sgx_seal_data`.
7. The Enclave releases the secret to perform functions.

Deleting a Policy

1. The Enclave follows the process of updating the Replay-Protected Activity Log to set the use counter to the maximum number of uses allowed, before releasing the secret for the last time.
2. User connects to the network to upload the Activity Log and receives a new secret.
3. The Enclave deletes the activity Log and the associated Monotonic Counter using `sgx_destroy_monotonic_counter`. If it is blocked by the attacker, the associated activity Log does not allow releasing of the secret as the secret inside the activity Log is invalidated or expired.

Time Based Policy

The sample code demonstrates a proper implementation of a Time-Based Policy in the form of an offline Digital Rights Management (DRM) Key that expires after a certain period of time. The sample code will not implement full DRM functionality. Instead, it demonstrates:

- Creating offline sealed data with the DRM key, a time stamp and the expiration policy.
- Verifying the DRM key has not expired before releasing the key to perform function.

Initializing a Policy

1. The Enclave retrieves the time reference and the time source nonce using `sgx_get_trusted_time`.
2. The Enclave fills the policy structure with the sample usage secret, the time policy, the time reference and the time source nonce returned by `sgx_get_trusted_time`.
3. The Enclave seals the policy structure into Sealed Data using `sgx_seal_data`.

Verifying a Policy

1. The Enclave verifies and decrypts the sealed data using `sgx_unseal_data`.
2. The Enclave retrieves the current time using `sgx_get_trusted_time`.
3. The Enclave verifies the time source nonce returned by `sgx_get_trusted_time` is the same as the time source nonce in the policy structure. If not, abort the operation.
4. Calculate time elapsed.
5. Verify the policy. If the time limit has expired, abort the operation.
6. The Enclave releases the secret to perform functions.

Reference Code for X509 Certificate Verification

The sample code shows how to use the open source OpenSSL* Library together with the Intel® Software Guard Extensions to verify an X509 certificate safely.

You can find the zip file `x509.zip` shipped together with the Intel® Software Guard Extensions Evaluation SDK in the `$(SGXSDKInstallPath)src\X509Verifier` folder. You only need to open the project on Microsoft* Visual Studio 2012 to compile it.

NOTE:

If the sample project is located in a system directory, administrator privilege is required to open it. You can copy the project folder to your directory if administrator permission cannot be granted.

The sample code contains a topenssl folder which is a slightly modified OpenSSL* source code that works together with the Intel® Software Guard Extensions Evaluation SDK. Only the code under `topenssl/crypto` has been ported and all the modifications are delimited by the macro `OPENSSL_FOR_SGX`.

The sample code contains an untrusted component and a trusted enclave. The topenssl code is treated as a static enclave library which is linked to the enclave. The flow of the untrusted component in the sample code is as follows::

1. Loads the enclave with `sgx_create_enclave` and reads an input certificate file.
2. ECALLs into the enclave to verify the input certificate chain with X509 functions provided by the topenssl library.
3. Destroys the enclave with `sgx_destroy_enclave`, recycles resources and exits.

You can find a valid certificate and an invalid certificate under `test_vrfcert/data` in the sample folder.

To concentrate on X509 certificate verification, the X509 sample code assumes the enclave file is locatedin the current working directory and doesn't store the launch token. See Sample Enclave for how to handle the enclave file path and store the launch token properly.

Verify X509 Certificate Chain

A trusted root Certificate Authority (CA) is a must before verifying a certificate chain. In the sample, a prepared root CA is hard-coded into the enclave’s code. Intel® SGX provides two sample certificate chains which has been signed by correspondent private key of the root CA under data folder.

You can use tool like openssl* to create your own root CA and replace the hard-coded root CA (contains public key only). After that you can create your own test certificate chain by yourself (to sign it by the private key of the root CA).

The certificate verification includes verification for certificate and Certificate Revocation List (CRL). The sample verifies the certificate chain with the X509 function `X509_verify_cert`.

NOTE:

The enclave file can be disassembled, so the algorithms used by the enclave developer will not remain secret such as private key of root CA.

You should confirm the integrity of the root CA and ensure the root CA can be trusted before using it within enclave.

Library Functions and Type Reference

This topic includes the following sub-topics to describe library functions and type reference for Intel® Software Guard Extensions Evaluation SDK:

- Untrusted Library Functions
- Trusted Libraries
- Function Descriptions
- Types and Enumerations
- Error Codes

Untrusted Library Functions

The untrusted library functions can only be called from application code - outside the enclave.

Enclave Creation and Destruction

These functions are used to either create or destroy enclaves:

- sgx_create_enclave
- sgx_destroy_enclave

Enclave Enumeration

Use this function to enumerate all the processes that have created and are currently using one or more enclaves. You can also use this function to obtain information about all enclaves loaded on the platform.

- sgx_enum_enclaves

Quoting Functions

These functions allow application enclaves to ensure that they are running on an Intel® Software Guard Extensions environment.

- sgx_init_quote
- sgx_get_quote_size
- sgx_get_quote
- sgx_report_attestation_status

Key Exchange Functions

These functions allow exchanging of secrets between ISV's server and enclaves. They are used in concert with the trusted Key Exchange functions.

- sgx_ra_get_msg1
- sgx_ra_proc_msg2

Platform Service Function

This function helps ISVs determine what Intel® SGX Platform Services are supported by the platform.

- sgx_get_ps_cap

Intel® SGX Enabling and Launch Control Functions

The enabling and launch control function helps you to enable the Intel® SGX device and return appropriate status.

- sgx_enable_device
- sgx_cap_enable_device

This function provides an Enclave Signing Key White List Certificate Chain. An Enclave Signing Key White List Certificate Chain contains the signing key(s) of the Intel® SGX application enclave(s) allowed to be launched. If the system has not acquired an up-to-date Enclave Signing Key White List Certificate Chain, you can provide the chain to the system by setting `sgx_register_wl_cert_chain`.

- sgx_register_wl_cert_chain

Intel® SGX device capability Functions

The SGX device capability function helps you to check if the client platform is enabled for Intel SGX or the software control interface is available to configure the Intel® SGX device.

- sgx_is_capable

Trusted Libraries

The trusted libraries are static libraries that link with the enclave binary. The Intel® Software Guard Extensions Evaluation SDK ships with several trusted libraries that cover domains such as standard C/C++ libraries, synchronization, encryption and more.

These functions/objects can only be used from within the enclave.

CAUTION:

Do not link the enclave with any untrusted library including C/C++ standard libraries. This action will either fail the enclave signing process or cause a runtime failure due to the use of restricted instructions.

Trusted Runtime System

The Intel® SGX trusted runtime system (tRTS) is a key component of the Intel® Software Guard Extensions Evaluation SDK. It provides the enclave entry point logic as well as other functions to be used by enclave developers.

- Intel® Software Guard Extensions Helper Functions
- Custom Exception Handling
- Debug API inside Trusted Libraries
- Intrinsic Functions

Intel® Software Guard Extensions Helper Functions

The tRTS provides the following helper functions for you to determine whether a given address is within or outside enclave memory.

- sgx_is_within_enclave
- sgx_is_outside_enclave

The tRTS provides a wrapper to the RDRAND instruction to generate a true random number from hardware. The C/C++ standard library functions `rand` and `srand` functions are not supported within an enclave because they only provide pseudo random numbers. Instead, enclave developers should use the `sgx_read_rand` function to get true random numbers.

- sgx_read_rand

Custom Exception Handling

The Intel® Software Guard Extensions Evaluation SDK supports exception handling with a Vector Exception Handling like API. You can write your own code to handle a limited set of hardware exceptions. For example, a CPUID instruction inside an Enclave will effectively result in a #UD fault (Invalid Opcode Exception). ISV enclave code can provide an exception handler to prevent the enclave from being trapped in an exception condition.

NOTE:

Custom exception handling is only supported in HW mode. Although the exception handlers can be registered in simulation mode, the exceptions cannot be caught and handled within the enclave.

NOTE:

OCALLs are not allowed in the exception hander.

The Custom Exception Handling APIs are listed below:

- sgx_register_exception_handler
- sgx_unregister_exception_handler

Custom Exception Handler for CPUID Instruction

If an ISV requires the use of the CPUID information within an enclave, then the enclave code must make an OCALL to perform the CPUID instruction in the untrusted application. The ISV could also leverage the intrinsics `__cpuid` and `__cpuidex`, or the functions `sgx_cpuid` and `sgx_cpuid_ex`,which the `sgx_tstdc` library provides, to which the instrinsics map. `sgx_cpuid` and `sgx_cpuid_ex` make an OCALL to the uRTS library to obtain CPUID data. However, in either case the ISV should be cognizant that the returned results are from the untrusted application. Thus it is recommended that threat evaluation is performed to ensure that comprised CPUID return values are not problematic. Ideally, sanity checking of the return values should be performed.

If an ISV's enclave uses a third party library which executes the CPUID instruction, then the ISV would need to provide a custom exception handler under the assumption that the third party library has not provided CPUID support. The ISV is responsible for analyzing the usage of the specific CPUID result provided by the untrusted domain to ensure it does not compromise the enclave security properties. Recommended implementation of the CPUID exception handler involves:

1. ISV analyzes the third party library CPUID usages, identifying required CPUID results.
2. ISV enclave code initialization routine populates a "cache" of the required CPUID results inside the enclave. This "cache" might be maintained by the RTS or by ISV code.
3. ISV enclave code initialization routine registers a custom exception handler.
4. The custom exception handler, when invoked, examines the exception information and faulting instruction. If the exception is caused by a CPUID instruction:
    a. Retrieve the "cached" CPUID result and populate the CPUID instruction output registers.
    b. Advance the RIP to bypass the CPUID instruction and complete the exception handling.

Debug API inside Trusted Libraries

You can use the following debug APIs inside enclave:

- IsDebuggerPresent
- OutputDebugString
- DebugBreak

Intrinsic Functions

The majority of Microsoft* Visual C++ intrinsics can be called inside the enclave, and an enclave project can include Microsoft* standard `<intrin.h>` directly with few restrictions. For example, you should not use intrinsics that generate instructions unsupported inside an enclave. All unsupported intrinsic functions generally fall into following categories:

- I/O related functions.
- Instructions requiring ring 0 privileges or can change privilege level.
- OS or system related functions.
- Intrinsics which are considered unprotected and encryption alternatives.

1. There are few requirements for including Microsoft* standard `<intrin.h>`:
    a. Add `$(VCInstallDir)include;$(IncludePath)` to `Include Directories`.
    b. Set `Ignore Standard Include Path` to `No`.
2. Use /Oi or #pragma intrinsic(…) to enable MSVC intrinsics.

The `<sgx_intrin.h>` also provides compile warnings for unsupported intrinsics.

Trusted Service Library

The Intel® Software Guard Extensions Evaluation SDK provides a trusted library named `sgx_tservice` for secure data manipulation and protection. The `sgx_tservice` library provides the following trusted functionality and services:

- Intel® Software Guard Extensions Instruction Wrapper Functions
- Intel® Software Guard Extensions Sealing and Unsealing Functions
- Platform Service Function
- Diffie–Hellman (DH) Session Establishment Functions

Intel® Software Guard Extensions Instruction Wrapper Functions

The `sgx_tservice` library provides functions for getting specific keys and for creating and verifying an enclave report. The API functions are listed below:

- sgx_get_key
- sgx_create_report
- sgx_verify_report

Intel® Software Guard Extensions Sealing and Unsealing Functions

The sgx_tservice library exposes APIs to create sealed data which is both confidentiality and integrity protected, and an API to unseal sealed data inside the enclave.

- sgx_seal_data
- sgx_seal_data_ex
- sgx_unseal_data

The library also provides APIs to help calculate the sealed data size, encrypt text length, and Message Authentication Code (MAC) text length.

- sgx_calc_sealed_data_size
- sgx_get_add_mac_txt_len
- sgx_get_encrypt_txt_len

Enclave Secret Sealing Introduction

When an enclave is instantiated, it provides protections (confidentiality and integrity) to the data by keeping it within the boundary of the enclave. Enclave developers should identify enclave data and/or state that is considered secret and potentially needs preservation across the following enclave destruction events:

- Application is done with the enclave and closes it.
- Application itself is closed.
- The platform is hibernated or shutdown.

In general, the secrets provisioned within an enclave are lost when the enclave is closed. However if the secret data needs to be preserved during one of these events for future use within an enclave, it must be stored outside the enclave boundary before closing the enclave. In order to protect and preserve the data, a mechanism is in place which allows enclave software to retrieve a key unique to that enclave. This key can only be generated by that enclave on that particular platform. Enclave software uses that key to encrypt data to the platform or to decrypt data already on the platform. Refer to these "encrypt" and "decrypt" operations as "sealing" and "unsealing" respectively as the data is cryptographically sealed to the enclave and platform.

To provide strong protection against potential key-wear-out attacks, a unique seal key is generated for each data blob encrypted with the `sgx_seal_data` API call. A key ID for each encrypted data blob is stored in clear alongside the encrypted data blob. The key ID is used to re-generate the seal key to decrypt the data blob.

AES-GCM (AES – Advanced Encryption Standard) is utilized to encrypt and MAC-protect the payload. To protect against software-based side channel attacks, the crypto implementation of AES-GCM utilizes AES-NI, which is immune to software-based side channel attacks. The Galois/Counter Mode (GCM) is a mode of operation of the AES algorithm. GCM assures authenticity of the confidential data (of up to about 64 GB per invocation) using a universal hash function. GCM can also provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted.GCM can also provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted. If the GCM input contains only data that is not to be encrypted, the resulting specialization of GCM, called GMAC (Galois Message Authentication Code), is simply an authentication mode for the input data. The `sgx_mac_aadata` API call restricts the input to non-confidential data to provide data origin authentication only. The single output of this function is the authentication tag.

Example Use Cases

One example is that an application may start collecting secret state while executing that needs to be preserved and utilized on future invocations of that application. Another example is during application installation, a secret key may need to be preserved and verified upon starting the application.

For these cases the seal APIs can be utilized to seal the secret data (key or state) in the examples above, and then unseal the secret data when needed.

Sealing

1. Use `sgx_calc_sealed_data_size` to calculate the number of bytes to allocate for the `sgx_sealed_data_t` structure.
2. Allocate memory for the `sgx_sealed_data_t` structure.
3. Call `sgx_seal_data` to perform sealing operation
4. Save the sealed data structure for future use.

Unsealing

1. Use `sgx_get_encrypt_txt_len` and `sgx_get_add_mac_txt_len` to determine the size of the buffers to allocate in terms of bytes.
2. Allocate memory for the decrypted text and additional text buffers.
3. Call `sgx_unseal_data` to perform the unsealing operation.

Platform Service Functions

The `sgx_tservice` library provides the following functions that allow an ISV to use platform services and get platform services security property.

- sgx_create_pse_session
- sgx_close_pse_session
- sgx_get_ps_sec_prop
- sgx_get_trusted_time
- sgx_create_monotonic_counter_ex
- sgx_create_monotonic_counter
- sgx_destroy_monotonic_counter
- sgx_increment_monotonic_counter
- sgx_read_monotonic_counter

Diffie–Hellman (DH) Session Establishment Functions

These functions allow an ISV to establish secure session between two enclaves using the EC DH Key exchange protocol.

- sgx_dh_init_session
- sgx_dh_responder_gen_msg1
- sgx_dh_initiator_proc_msg1
- sgx_dh_responder_proc_msg2
- sgx_dh_initiator_proc_msg3

C Standard Library

The Intel® Software Guard Extensions Evaluation SDK includes a trusted version of the C standard library. The library is named `sgx_tstdc` (trusted standard C), and can only be used inside an enclave. Standard C headers are located under `$(SGXSDKInstallPath)include\tlibc`.

`sgx_tstdc` provides a subset of C99 functions that are ported from OpenBSD* project. Some functions are not allowed to use inside the enclave for following reasons:

- The definition implies usage of a restricted CPU instruction.
- The definition is known to be unsafe or insecure.
- The definition implementation is too large to fit inside an enclave or relies heavily on information from the untrusted domain.
- The definition is compiler specific, and not part of the standard.
- The definition is a part of the standard, but it is not supported by a specific compiler.

See Unsupported C Standard Functions for a list of unsupported C99 definitions within an enclave.

Locale Functions

A trusted version of locale functions is not provided primarily due to the size restriction. Those functions rely heavily on the localization data (normally 1MB to 2MB), which should be preloaded into the enclave in advance to ensure that it will not be modified from the untrusted domain. This practice would increase the footprint of an enclave, especially for those enclaves not depending on the locale functionality. Moreover, since localization data is not available, wide character functions inquiring enclave locale settings are not supported either.

Random Number Generation Functions

The random functions `srand` and `rand` are not supported in the Intel® SGX SDK C library. A true random function `sgx_read_rand` is provided in the tRTS library by using the RDRAND instruction. However, in the Intel® SGX simulation environment, this function still generates pseudo random numbers because RDRAND may not be available on the hardware platform.

String Functions

The functions `strcpy` and `strcat` are not supported in the Intel® SGX SDK C library. You are recommended to use `strncpy` and `strncat` instead.

Abort Function

The `abort()` function is supported within an enclave but has different behavior. When a thread calls the abort function, it makes the enclave unusable by setting the enclave state to a specific value that allows the tRTS and application to detect and report this event. The aborting thread generates an exception and exits the enclave, while other enclave threads continue running normally until they exit the enclave. Once the enclave is in the unusable state, subsequent enclave calls and OCALL returns generate the same error indicating that the enclave is no longer usable. After all thread calls abort, the enclave is locked and cannot be recovered. You have to destroy, reload and reinitialize the enclave to make it usable again.

Thread Synchronization Primitives

Multiple untrusted threads may enter an enclave simultaneously as long as more than one thread context is defined by the application and created by the untrusted loader. Once multiple threads execute concurrently within an enclave, they will need some forms of synchronization mechanism if they intend to operate on any global data structure. In some cases, threads may use the atomic operations provided by the processor’s ISA. In the general case, however, they would use synchronization objects and mechanisms similar to those available outside the enclave.

The Intel® Software Guard Extensions Evaluation SDK already supports mutex and conditional variable synchronization mechanisms by means of the following API and data types defined in the

Types and Enumerations section. Some functions included in the trusted Thread Synchronization library may make calls outside the enclave (OCALLs). Developers who use these APIs must first import needed OCALL functions from the `sgx_tstdc.edl` file. Otherwise, developers will get a linker error when the enclave is being built; see Calling Functions outside the Enclave for additional details. The table below illustrates the primitives that the SGX Thread Synchronization library supports, as well as the OCALLs that each API function needs.

	Function API	OCall Function
Mutex Synchronization	sgx_thread_mutex_init	
	sgx_thread_mutex_destroy	
	sgx_thread_mutex_lock	sgx_thread_wait_untrusted_event_ocall
	sgx_thread_mutex_trylock	
	sgx_thread_mutex_unlock	sgx_thread_set_untrusted_event_ocall
Condition Variable Synchronization	sgx_thread_cond_init	
	sgx_thread_cond_destroy	
	sgx_thread_cond_wait	sgx_thread_wait_untrusted_event_ocall sgx_thread_setwait_untrusted_events_ocall
	sgx_thread_cond_signal	sgx_thread_set_untrusted_event_ocall
	sgx_thread_cond_broadcast	sgx_thread_set_multiple_untrusted_events_ocall
Thread Management	sgx_thread_self	

Query CPUID inside Enclave

The Intel® Software Guard Extensions Evaluation SDK provides two functions for enclave developers to query a subset of CPUID information inside the enclave:

- sgx_cpuid
- sgx_cpuidex

C++ Language Support

The Intel® Software Guard Extensions Evaluation SDK provides a trusted library for C++ support inside the enclave. C++ developers would utilize advanced C++ features that require C++ runtime libraries.

The ISO/IEC 14882:2003 C++ standard is chosen as the baseline for the Intel® Software Guard Extensions Evaluation SDK trusted library. Most of standard C++ features are fully supported inside the enclave, and including:

1. Dynamic memory management with new/delete;
2. Global initializers are supported (usually used in the construction of global objects);
3. Run-time Type Identification (RTTI);
4. C++ exception handling inside the enclave.

Currently, global destructors are not supported due to the reason that EPC memory will be recycled when destroying an enclave.

NOTE

C++ objects are not supported in enclave interface definitions. If an application needs to pass a C++ object across the enclave boundary, recommended implementation is to store the C++ object's data in a C struct and marshal the data across the enclave interface. Then you need to instantiate the C++ object in the other domain with the marshaled 'C' struct passed in to the constructor (or you may update existing instantiated objects with appropriate operators).

C++ Standard Library

The Intel® Software Guard Extensions Evaluation SDK includes a trusted version of the C++ standard library (including STL) that conforms to the C++03 standard. The library is ported from STLport. As STLport or other open source implementations of higher C++ standard come available, C++11 support may be added later.

As for C++ standard library, most functions will work just as its untrusted part, but here is a high level summary of features that are supported inside the enclave:

1. I/O related functions and classes, like `<iostream>`;
2. Functions depend on locale library;
3. Any other functions that require system calls.

Furthermore, C functions can be used as the language for trusted and untrusted interfaces. While you can use C++ to develop your enclaves, you should not pass C++ objects across the enclave boundary.

Cryptography Library

The Intel® Software Guard Extensions Evaluation SDK includes a trusted cryptography library named `sgx_tcrypto`. It includes the cryptographic functions used by other trusted libraries included in the SDK, such as the `sgx_tservice` library. Thus, the functionality provided by this library might be somewhat limited. If you need additional cryptographic functionality, you would have to develop your own trusted cryptographic library.

- sgx_sha256_msg
- sgx_sha256_init
- sgx_sha256_update
- sgx_sha256_get_hash

- sgx_sha256_close
- sgx_rijndael128GCM_encrypt
- sgx_rijndael128GCM_decrypt
- sgx_rijndael128_cmac_msg
- sgx_cmac128_init
- sgx_cmac128_update
- sgx_cmac128_final
- sgx_cmac128_close
- sgx_aes_ctr_encrypt
- sgx_aes_ctr_decrypt
- sgx_ecc256_open_context
- sgx_ecc256_close_context
- sgx_ecc256_create_key_pair
- sgx_ecc256_compute_shared_dhkey
- sgx_ecc256_check_point
- sgx_ecdsa_sign
- sgx_ecdsa_verify

Key Exchange Functions

These functions allow an ISV to exchange secrets between its server and its enclaves. They are used in concert with untrusted Key Exchange functions.

- sgx_ra_init
- sgx_ra_get_keys
- sgx_ra_close

Function Descriptions

This topic describes various functions including their syntax, parameters, return values, and requirements.

NOTE

When an API function lists an EDL in its requirements, users need to explicitly import such library EDL file in their enclave's EDL.

sgx_create_enclave

Loads the enclave using its file name and initializes it using a launch token.

`sgx_create_enclave` is a macro for the `sgx_create_enclavea`(ANSI) or `sgx_create_enclavew` (Unicode) function.

The compiler will use the Unicode version if UNICODE is defined in the project.

Syntax

```
#if !defined(NDEBUG) || defined(EDEBUG)
    #define SGX_DEBUG_FLAG ((int)1)
#else
    #define SGX_DEBUG_FLAG ((int)0)
#endif
sgx_status_t sgx_create_enclave(
```

```
    const char *file_name,
    const int debug,
    sgx_launch_token_t *launch_token,
    int *launch_token_updated,
    sgx_enclave_id_t *enclave_id,
    sgx_misc_attribute_t *misc_attr
);
```

Parameters

file_name [in]

Name or full path to the enclave image. This parameter is identical to the **lpFileName** parameter in **CreateFile()**. If the project is using Unicode character set, `file_name` should be an Unicode string. If the project is using Multi-Byte character set, **file_name** should be an ANSI string.

debug [in]

The valid value is 0 or 1.

0 indicates to create the enclave in non-debug mode and 1 indicates to create the enclave in debug mode. An enclave created in non-debug mode cannot be debugged. The code/data memory inside an enclave created in debug mode is accessible by the debugger or other software outside of the enclave and thus is *not* under the same memory access protections as an enclave created in non-debug mode. Enclaves should only be created in debug mode for debug purposes. A helper macro `SGX_DEBUG_FLAG` is provided to create an enclave in debug mode when EDEBUG is defined or NDEBUG is not defined.

launch_token [in/out]

A pointer to an sgx_launch_token_t object used to initialize the enclave to be created. Must not be NULL. The caller can provide an all-0 buffer as the sgx_launch_token_t object, in which case, the function will attempt to create a valid sgx_launch_token_tobject and store it in the buffer. The caller should store the sgx_launch_token_t object and re-use it in future calls to create the same enclave. Certain platform configuration changes can invalidate a previously stored sgx_launch_token_t object. If the token provided is *not* valid, the function will attempt to update it to a valid one.

launch_token_updated [out]

The output is 0 or 1. 0 indicates the launch token has not been updated. 1 indicates the launch token has been updated.

enclave_id [out]

A pointer to an sgx_enclave_id_t that receives the enclave ID or handle. Must not be NULL.

misc_attr [out, optional]

A pointer to an sgx_misc_attribute_t structure that receives the misc select and attributes of the enclave. This pointer may be NULL if the information is not needed.

Return value

SGX_SUCCESS

The enclave was loaded and initialized successfully.

SGX_ERROR_INVALID_ENCLAVE

The enclave file is corrupted.

SGX_ERROR_INVALID_PARAMETER

The 'enclave_id', 'updated' or 'token' parameter is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory available to complete `sgx_create_enclave()`.

SGX_ERROR_ENCLAVE_FILE_ACCESS

The enclave file can't be opened. It may be caused by enclave file not being found or no privilege to access the enclave file.

SGX_ERROR_INVALID_METADATA

The metadata embedded within the enclave image is corrupt or missing.

SGX_ERROR_INVALID_VERSION

The enclave metadata version (created by the signing tool) and the untrusted library version (uRTS) do not match.

SGX_ERROR_INVALID_SIGNATURE

The signature for the enclave is not valid.

SGX_ERROR_OUT_OF_EPC

The protected memory has run out. For example, a user is creating too many enclaves, the enclave requires too much memory, or we cannot load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_NO_DEVICE

The SGX device is not valid. This may be caused by the SGX driver not being installed or the SGX driver being disabled.

SGX_ERROR_MEMORY_MAP_CONFLICT

During enclave creation, there is a race condition for mapping memory between the loader and another thread. The loader may fail to map virtual address. If this error code is encountered, create the enclave again.

SGX_ERROR_DEVICE_BUSY

The SGX driver or low level system is busy when creating the enclave. If this error code is encountered, we suggest creating the enclave again.

SGX_ERROR_MODE_INCOMPATIBLE

The target enclave mode is incompatible with the mode of the current RTS. For example, a 64-bit application tries to load a 32-bit enclave or a simulation uRTS tries to load a hardware enclave.

SGX_ERROR_SERVICE_UNAVAILABLE

`sgx_create_enclave()` needs the AE service to get a launch token. If the service is not available, the enclave may not be launched.

SGX_ERROR_SERVICE_TIMEOUT

The request to the AE service timed out.

SGX_ERROR_SERVICE_INVALID_PRIVILEGE

The request requires some special attributes for the enclave, but is not privileged.

SGX_ERROR_NDEBUG_ENCLAVE

The enclave is signed as a product enclave and cannot be created as a debuggable enclave.

SGX_ERROR_UNDEFINED_SYMBOL

The enclave contains an import table.

The signing tool should typically report this type of error when the enclave is built.

SGX_ERROR_INVALID_MISC

The MiscSelct/MiscMask settings are not correct.

SGX_ERROR_VMM_INCOMPATIBLE

The virtual machine monitor is not compatible.

SGX_ERROR_HYPERV_ENABLED

Incompatible versions of Windows* 10 OS and Hyper-V* are detected. In this case, you need to disable Hyper-V on the target machine.

SGX_ERROR_UNEXPECTED

An unexpected error is detected.

Description

The `sgx_create_enclave` function will load and initialize the enclave using the enclave file name and a launch token. If the launch token is incorrect, it will get a new one and save it back to the input parameter "token", and the parameter "updated" will indicate that the launch token was updated.

If both enclave and license are valid, the function will return a value of `SGX_SUCCESS`. The enclave ID (handle) is returned via the `enclave_id` parameter.

The library `sgx_urts.lib` provides this function to load an enclave with Intel® SGX hardware. This function cannot be used to load an enclave linked with the simulation library. On the other hand, the simulation library `sgx_urts_sim.lib` exposes an identical interface which can only load a simulative enclave. Running in simulation mode does not require Intel® SGX hardware/driver. However, it does not provide hardware protection.

The randomization of the load address of the enclave is dependent on the operating system. The address of the heap and stack is not randomized and is at a constant offset from the enclave base address. Different versions of Windows may randomize or not randomize the base address differently. A compromised loader or operating system (both of which are outside the TCB) can remove the randomization entirely.

NOTE

The enclave writer should not rely on the randomization of the base address of the enclave.

Requirements

Header	`sgx_urts.h`
Library	`sgx_urts.lib` or `sgx_urts_sim.lib` (simulation)

sgx_destroy_enclave

The `sgx_destroy_enclave` function destroys an enclave and frees its associated resources.

Syntax

```
sgx_status_t sgx_destroy_enclave(
```

```
    const sgx_enclave_id_t enclave_id
);
```

Parameters

enclave_id [in]

An enclave ID or handle that was generated by sgx_create_enclave.

Return value

SGX_SUCCESS

The enclave was unloaded successfully.

SGX_ERROR_INVALID_ENCLAVE_ID

The enclave ID (handle) is not valid. The enclave has not been loaded or the enclave has already been destroyed.

Description

The `sgx_destroy_enclave` function destroys an enclave and releases its associated resources and invalidates the enclave ID or handle.

The function will block until no other threads are executing inside the enclave.

It is highly recommended that the `sgx_destroy_enclave` function be called after the application has finished using the enclave to avoid possible deadlocks.

The library `sgx_urts.lib` exposes this function to destroy a previously created enclave in hardware mode, while `sgx_urts_sim.lib` provides a simulative counterpart.

See more details in Load and Unload an Enclave.

Requirements

Header	`sgx_urts.h`
Library	`sgx_urts.lib` or `sgx_urts_sim.lib` (simulation)

sgx_enum_enclaves

Enumerates the process IDs that have loaded enclaves, as well as their corresponding enclave IDs and enclave size.

Syntax

```
sgx_status_t sgx_enum_enclaves(
    EnclaveEnumArrayType* pEnclaveEnum,
    DWORD cb,
    DWORD* pBytesNeeded;
);
```

Parameters

pEnclaveEnum [out]

Pointer to an array of structures containing process ID, enclave ID and enclave size.

cb [in]

Size allocated for the array of structures of type `EnclaveEnumArrayType`.

pBytesNeeded [out]

The number of bytes required to store the complete array of structures of type `EnclaveEnumArrayType`.

Return value

SGX_ERROR_FEATURE_NOT_SUPPORTED

This API has been deprecated and is not longer supported.

Description

The `sgx_enum_enclaves` function has been deprecated and is not longer supported.

Requirements

Header	`sgx_urts.h`
Library	`sgx_urts.lib`

sgx_init_quote

`sgx_init_quote` returns information needed by an Intel® SGX application to get a quote of one of its enclaves.

Syntax

```
sgx_status_t sgx_init_quote(
    sgx_target_info_t *p_target_info,
    sgx_epid_group_id_t *p_gid
);
```

Parameters

p_target_info [out]

Allows an enclave for which the quote is being created, to create report that only QE can verify.

p_gid [out]

ID of platform's current EPID group.

Return value

SGX_SUCCESS

All of the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers are invalid.

SGX_ERROR_AE_INVALID_EPIDBLOB

The EPID blob is corrupted.

SGX_ERROR_BUSY

The requested service is temporarily not available

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UPDATE_NEEDED

Intel® SGX needs to be updated.

SGX_ERROR_UNEXPECTED

An unexpected error was detected.

Description

Calling `sgx_init_quote` is the first thing an Intel® Software Guard Extensions application does in the process of getting a quote of an enclave. The content of `p_target_info` changes when the QE changes. The content of p_gid changes when the platform SVN changes.

It's suggested that the caller should wait (typically several seconds to tens of seconds) and retry this API if **SGX_ERROR_BUSY** is returned.

Requirements

Header	`sgx_uae_service.h`
Library	`sgx_uae_service.lib` or `sgx_uae_service_sim.lib` (simulation)

sgx_get_quote_size

`sgx_get_quote_size` returns the required buffer size for the quote.

Syntax

```
sgx_status_t sgx_get_quote_size(
    const uint8_t *p_sig_rl,
    uint32_t *p_quote_size
);
```

Parameters

p_sig_rl [in]

Optional revoke list of signatures, can be NULL.

p_quote_size [out]

Indicate the size of quote buffer.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

The `p_quote_size` pointer is invalid or the other input parameters are corrupted.

Description

You cannot allocate a chunk of memory at compile time because the size of the quote is not a fixed value. Instead, before trying to call `sgx_get_quote`, call `sgx_get_quote_size` first to get the buffer size and then allocate enough memory for the quote.

Requirements

Header	`sgx_uae_service.h`
Library	`sgx_uae_service.lib` or `sgx_uae_service_sim.lib` (simulation)

sgx_get_quote

`sgx_get_quote` generates a linkable or un-linkable QUOTE.

Syntax

```
sgx_status_t sgx_get_quote(
     const sgx_report_t *p_report,
     sgx_quote_sign_type_t quote_type,
     const sgx_spid_t *p_spid,
     const sgx_quote_nonce_t *p_nonce,
     const uint8_t *p_sig_rl,
     uint32_t sig_rl_size,
     sgx_report_t *p_qe_report,
     sgx_quote_t *p_quote,
     uint32_t quote_size
);
```

Parameters

p_report [in]

Report of enclave for which quote is being calculated.

quote_type [in]

`SGX_UNLINKABLE_SIGNATURE` for unlinkable quote or `SGX_LINKABLE_SIGNATURE` for linkable quote.

p_spid [in]

ID of service provider.

p_nonce [in]

Optional nonce, if `p_qe_report` is not NULL, then nonce should not be NULL as well.

p_sig_rl [in]

Optional revoke list of signatures, can be NULL.

sig_rl_size [in]

Size of `p_sig_rl`, in bytes. If the `p_sig_rl` is NULL, then `sig_rl_size` shall be 0.

p_qe_report [out]

Optional output. If not NULL, report of QE target to the calling enclave will be copied to this buffer, and in this case, nonce should not be NULL as well.

p_quote [out]

The major output of `get_quote`, the quote itself, linkable or unlinkable depending on `quote_type` input. quote cannot be NULL.

quote_size [in]

Indicates the size of the quote buffer. To get the size, user shall call `sgx_get_quote_size` first.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers are invalid.

SGX_ERROR_AE_INVALID_EPIDBLOB

The EPID blob is corrupted.

SGX_ERROR_EPID_MEMBER_REVOKED

The EPID group membership has been revoked. The platform is not trusted. Updating the platform and retrying will not remedy the revocation.

SGX_ERROR_BUSY

The requested service is temporarily not available.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond.

SGX_ERROR_SERVICE_TIMEOUT

A request to AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UPDATE_NEEDED

Intel® SGX needs to be updated.

SGX_ERROR_UNEXPECTED

An unexpected error was detected.

Description

Both EPID Member and Verifier need to know the Group Public Key and the EPID Parameters used. These values not being returned by either `sgx_init_quote()` or `sgx_get_quote()` reflects the reliance on the Intel® Attestation Service (IAS). With the IAS in place, simply sending the GID to the IAS (through the Intel® SGX application and PS) is sufficient for the IAS to know which public key and parameters to use.

It's suggested that the caller should wait (typically several seconds to tens of seconds) and retry this API if **SGX_ERROR_BUSY** is returned.

Requirements

Header	`sgx_uae_service.h`
Library	`sgx_uae_service.lib` or `sgx_uae_service_sim.lib` (simulation)

sgx_ra_get_msg1

`sgx_ra_get_msg1` is used to get the remote attestation and key exchange protocol message 1 to send to a service provider. The application enclave should use `sgx_ra_init` function to create the remote attestation and key exchange process context, and return to the untrusted code, before the untrusted code can invoke this function.

Syntax

```
sgx_status_t sgx_ra_get_msg1(
    sgx_ra_context_t context,
    sgx_enclave_id_t eid,
    sgx_ecall_get_ga_trusted_t p_get_ga,
    sgx_ra_msg1_t *p_msg1
);
```

Parameters

context [in]

Context returned by the `sgx_ra_init` function inside the application enclave.

eid [in]

ID of the application enclave which is going to be attested.

p_get_ga [in]

Function pointer of the ECALL proxy `sgx_ra_get_ga` generated by `sgx_edger8r`. The application enclave should link with `sgx_tkey_exchange` library and import `sgx_tkey_exchange.edl` in the enclave EDL file to expose the ECALL proxy for `sgx_ra_get_ga`.

p_msg1 [out]

Message 1 used by the remote attestation and key exchange protocol.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers are invalid.

SGX_ERROR_AE_INVALID_EPIDBLOB

The EPID blob is corrupted.

SGX_ERROR_EPID_MEMBER_REVOKED

The EPID group membership has been revoked. The platform is not trusted. Updating the platform and retrying will not remedy the revocation.

SGX_ERROR_BUSY

The requested service is temporarily not available.

SGX_ERROR_UPDATE_NEEDED

Intel® SGX needs to be updated.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond.

SGX_ERROR_SERVICE_TIMEOUT

A request to AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_INVALID_STATE

The API is invoked in incorrect order or state.

SGX_ERROR_UNEXPECTED

An unexpected error was detected.

Description

The application also passes in a pointer to the untrusted proxy corresponding to `sgx_ra_get_ga`, which is exposed by the trusted key exchange library. This reflects the fact that the names of untrusted proxies are enclave-specific.

It's suggested that the caller should wait (typically several seconds to tens of seconds) and retry this API if **SGX_ERROR_BUSY** is returned.

Requirements

Header	`sgx_ukey_exchange.h`
Library	`sgx_ukey_exchange.lib`or `sgx_ukey_exchangemt.lib`

sgx_ra_proc_msg2

`sgx_ra_get_msg2` is used to process the remote attestation and key exchange protocol message 2 from the service provider and generate message 3 to send to the service provider. If the service provider accepts message 3, negotiated session keys between the application enclave and the service provider are ready for use. The application enclave can use `sgx_ra_get_keys` function to retrieve the negotiated keys and can use `sgx_ra_close` function to release the context of the remote attestation and key exchange process. If processing message 2 results in an error, the application should notify the service provider of the error or the service provider needs a time-out mechanism to terminate the remote attestation transaction when it does not receive message 3.

Syntax

```
sgx_status_t sgx_ra_proc_msg2(
    sgx_ra_context_t context,
    sgx_enclave_id_t eid,
    sgx_ecall_proc_msg2_trusted_t p_proc_msg2,
    sgx_ecall_get_msg3_trusted_t p_get_msg3,
    const sgx_ra_msg2_t *p_msg2,
    uint32_t msg2_size,
    sgx_ra_msg3_t **pp_msg3,
    uint32_t *p_msg3_size
);
```

Parameters

context [in]

Context returned by `sgx_ra_init`.

eid [in]

ID of the application enclave which is going to be attested.

p_proc_msg2 [in]

Function pointer of the ECALL proxy `sgx_ra_proc_msg2_trusted_t` generated by `sgx_edger8r`. The application enclave should link with `sgx_tkey_exchange` library and import the `sgx_tkey_exchange.edl` in the EDL file of the application enclave to expose the ECALL proxy for `sgx_ra_get_msg2`.

p_get_msg3 [in]

Function pointer of the ECALL proxy `sgx_ra_get_msg3_trusted_t` generated by `sgx_edger8r`. The application enclave should link with `sgx_tkey_exchange` library and import the `sgx_tkey_exchange.edl` in the EDL file of the application enclave to expose the ECALL proxy for `sgx_ra_get_msg3`.

p_msg2 [in]

`sgx_ra_msg2_t` message 2 from the service provider received by application.

msg2_size [in]

The length of p_msg2 (in bytes).

pp_msg3 [out]

`sgx_ra_msg3_t` message 3 to be sent to the service provider. The message buffer is allocated by the `sgx_ukey_exchange` library. The caller should free the buffer after use.

p_msg3_size [out]

The length of pp_msg3 (in bytes).

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers are invalid.

SGX_ERROR_AE_INVALID_EPIDBLOB

The EPID blob is corrupted.

SGX_ERROR_EPID_MEMBER_REVOKED

The EPID group membership has been revoked. The platform is not trusted. Updating the platform and retrying will not remedy the revocation.

SGX_ERROR_BUSY

The requested service is temporarily not available.

SGX_ERROR_UPDATE_NEEDED

Intel® SGX needs to be updated.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond.

SGX_ERROR_SERVICE_TIMEOUT

A request to AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_INVALID_STATE

The API is invoked in incorrect order or state.

SGX_ERROR_INVALID_SIGNATURE

The signature is invalid.

SGX_ERROR_MAC_MISMATCH

Indicates verification error for reports, sealed data, etc.

SGX_ERROR_UNEXPECTED

An unexpected error was detected.

Description

The `sgx_ra_proc_msg2` processes the incoming message 2 and returns message 3. Message 3 is allocated by the library, so the caller should free it after use.

It's suggested that the caller should wait (typically several seconds to tens of seconds) and retry this API if **SGX_ERROR_BUSY** is returned.

Requirements

Header	`sgx_ukey_exchange.h`
Library	`sgx_ukey_exchange.lib` or `sgx_ukey_exchangemt.lib`

sgx_report_attestation_status

`sgx_report_attestation_status` reports information from Intel Attestation Server during a remote attestation to help to decide whether TCB update is required. It's recommended to always call `sgx_report_attestation_status` after a remote attestation, whether it succeeds or fails.

Syntax

```
sgx_status_t sgx_report_attestation_status (
    const sgx_platform_info_t* p_platform_info
    int attestation_status,
    sgx_update_info_bit_t* p_update_info
);
```

Parameters

p_platform_info [in]

Pointer to opaque structure received from Intel Attestation Server.

attestation_status [in]

The value indicates whether remote attestation succeeds or fails. If attestation succeeds, the value is 0. If it fails, the value will be others.

p_update_info [out]

Pointer to the buffer that receives the update information only when the return value of `sgx_report_attestation_status` is `SGX_ERROR_UPDATE_NEEDED`.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers are invalid.

SGX_ERROR_AE_INVALID_EPIDBLOB

The EPID blob is corrupted.

SGX_ERROR_EPID_MEMBER_REVOKED

The EPID group membership has been revoked. The platform is not trusted. Updating the platform and retrying will not remedy the revocation.

SGX_ERROR_UPDATE_NEEDED

Intel® SGX needs to be updated.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond.

SGX_ERROR_SERVICE_TIMEOUT

A request to AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UNEXPECTED

An unexpected error was detected.

Description

The application calls `sgx_report_attestation_status` after remote attestation to help to recover the TCB.

Requirements

Header	`sgx_uae_service.h`
Library	`sgx_uae_service.lib` or `sgx_uae_service_sim.lib` (simulation)

sgx_get_ps_cap

`sgx_get_ps_cap` returns the platform service capability of the platform.

Syntax

```
sgx_status_t sgx_get_ps_cap(
    sgx_ps_cap_t* p_sgx_ps_cap
);
```

Parameters

p_sgx_ps_cap [out]

A pointer to See "sgx_ps_cap_t" on page 206 structure indicates the platform service capability of the platform.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

The ps_cap pointer is invalid.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_UNEXPECTED

An unexpected error is detected.

Description

Before using Platform Services provided by the trusted Architecture Enclave support library, you need to call `sgx_get_ps_cap` first to get the capability of the platform.

Requirements

Header	`sgx_uae_service.h`
Library	`sgx_uae_service.lib` or `sgx_uae_service_sim.lib` (simulation)

sgx_register_wl_cert_chain

`sgx_register_wl_cert_chain` helps you to provide an Enclave Signing Key White List Certificate Chain. An Enclave Signing Key White List Certificate Chain contains the signing key(s) of the Intel® SGX application enclave(s). If the system has not acquired an up-to-date Enclave Signing Key White List Certificate Chain, you can provide the chain to the system by setting `sgx_register_wl_cert_chain`.

Syntax

```
sgx_status_t sgx_register_wl_cert_chain(
     const TCHAR *CertChainPath
);
```

Parameters

CertChainPath [in]

The full path of Enclave White List Cert Chain file.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

The `ps_cap` pointer is invalid.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond.

SGX_ERROR_UNEXPECTED

An unexpected error is detected.

Description

If you have an update-to-date Enclave Signing Key White List Certificate Chain, you need to call `sgx_register_wl_cert_chain`once first to launch enclaves.

Requirements

Header	`sgx_status.h or sgx_uae_service.h`
Library	`sgx_status.dll or sgx_uae_service.dll`

NOTE:

To avoid dependency issues, it's recommended to use `sgx_status.dll` for an application installer and `sgx_uae_service.dll` for an Intel SGX application.

sgx_enable_device

`sgx_enable_device` helps ISV applications to enable the Intel® SGX device and return appropriate status. If a reboot is required, ISV applications can decide whether to notify users of the restart requirement or not.

Syntax

```
sgx_status_t sgx_enable_device(
    sgx_device_status_t *sgx_device_status
);
```

Parameters

sgx_device_status [out]

The status of Intel SGX device.

SGX_ENABLED

Intel SGX device is already enabled

SGX_DISABLED_REBOOT_REQUIRED

Intel SGX device is currently disabled and a reboot is required to enable it.

SGX_DISABLED_LEGACY_OS

The operating system does not support enabling Intel SGX device

SGX_DISABLED

Intel SGX device is disabled

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

The `sgx_device_status` pointer is invalid.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond.

SGX_ERROR_VMM_INCOMPATIBLE

The virtual machine monitor is not compatible.

SGX_ERROR_HYPERV_ENABLED

The detected version of Windows* 10 is incompatible with Hyper-V*. In this case, you need to disable Hyper-V* on the target machine.

SGX_ERROR_UNEXPECTED

An unexpected error is detected.

Description

ISV applications can call `sgx_enable_device` to enable Intel SGX device dynamically.

A platform update may have occurred disabling SGX, and execution of this API will re-enable SGX but only after a reboot. If SGX is not currently enabled on the platform, the ISV application determines the next course of action:

a. Continue to run in non-SGX mode
b. Shut down the application and inform the user that a reboot is required before this application can run.

NOTE: In the case SGX_DISABLED is returned, manual BIOS configuration by the user may be required. The ISV needs to determine the recommended course of action to the user.

Requirements

Header	`sgx_uae_service.h`
Library	`sgx_uae_service.dll`

NOTE:

It's recommended to use `sgx_cap_enable_device` for an application installer and `sgx_enable_device` for an Intel SGX application.

sgx_cap_enable_device

`sgx_cap_enable_device` helps ISV application installers to enable the Intel® SGX device and return appropriate status. If a reboot is required, ISV application installers can decide whether to notify users of the restart requirement or not.

Syntax

```
sgx_status_t sgx_cap_enable_device(
    sgx_device_status_t *sgx_device_status
);
```

Parameters

sgx_device_status [out]

The status of the Intel SGX device.

SGX_ENABLED

Intel SGX device is already enabled.

SGX_DISABLED_REBOOT_REQUIRED

Intel SGX device is currently disabled and a reboot is required to enable it.

SGX_DISABLED_LEGACY_OS

The operating system does not support enabling of the Intel SGX device.

SGX_DISABLED

Intel SGX device is disabled.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

The `sgx_device_status` pointer is invalid.

SGX_ERROR_VMM_INCOMPATIBLE

The virtual machine monitor is not compatible.

SGX_ERROR_HYPERV_ENABLED

The detected version of Windows* 10 is incompatible with Hyper-V*. In this case, you need to disable Hyper-V* on the target machine.

SGX_ERROR_EFI_NOT_SUPPORTED

The operating system installed does not support the EFI interface.

SGX_ERROR_NO_PRIVILEGE

The application does not have the required privileges to enable Intel SGX. Run the application with the administrator privileges to enable the Intel SGX device.

SGX_ERROR_UNEXPECTED

An unexpected error is detected.

Description

ISV application installers can call `sgx_cap_enable_device` to enable the Intel SGX device dynamically.

ISV application installers can run this API before installing the SGX PSW to configure SGX on the client platform if it has not already been configured. (NOTE: The ability to dynamically enable/-configure SGX on a client platform is dependent on the availability of a SW Control Interface made available by the BIOS).

An application installer calling `sgx_cap_enable_device` must run with the administrator privileges to enable the Intel SGX device.

NOTE: In case SGX_DISABLED is returned, manual BIOS configuration by the user may be required. The ISV needs to determine the recommended course of action to the user.

Requirements

Header	`sgx_capable.h`
Library	`sgx_capable.dll`

NOTE:

It's recommended to use `sgx_cap_enable_device` for an application installer and `sgx_enable_device` for an Intel SGX application.

sgx_is_capable

`sgx_is_capable` helps ISV applications determine whether the client platform is capable of running Intel® SGX applications. Applications using this interface must run with administrator privilege to get the status successfully.

Syntax

```
sgx_status_t sgx_is_capable(
    int *sgx_capable
);
```

Parameters

sgx_capable [out]

Whether the platform is capable of running Intel® SGX applications.

1

The platform is enabled for Intel SGX or the Software Control Interface is available to configure the Intel SGX device.

0

The platform cannot be setup to run Intel SGX applications or the function call returned with an error.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

The `sgx_capable` pointer is invalid.

SGX_ERROR_EFI_NOT_SUPPORTED

The operating system installed does not support EFI interface.

SGX_ERROR_NO_PRIVILEGE

The application does not have the required privilege to read the EFI variable. Run the application with administrator privilege to query the Intel SGX device status.

SGX_ERROR_UNEXPECTED

An unexpected error is detected.

Description

The primary use of this API is by ISV application installers to determine whether a client platform supports the use of SGX. Based on a return value of '1', the application installer can proceed with the installation of the SGX PSW. The API `sgx_is_capable` is packaged within a small DLL `sgx_capable.dll`. Thus, the applications installed/updated via the cloud can use this API to first determine whether the client platform supports SGX before downloading the SGX PSW. This will save a lot of user's time, if SGX is not supported.

Requirements

Header	`sgx_capable.h`
Library	`sgx_capable.dll`

sgx_is_within_enclave

The `sgx_is_within_enclave` function checks that the buffer located at the pointer `addr` with its length of `size` is an address that is strictly within enclave address space.

Syntax

```
int sgx_is_within_enclave (
    const void *addr,
```

```
    size_t size
);
```

Parameters

addr [in]

The start address of the buffer.

size [in]

The size of the buffer.

Return value

1

The buffer is strictly within the enclave address space.

0

The whole buffer or part of the buffer is not within the enclave, or the buffer is wrapped around.

Description

`sgx_is_within_enclave` simply compares the start and end address of the buffer with the enclave address space. It does not check the property of the address. Given a function pointer, you sometimes need to confirm whether such a function is within the enclave. In this case, it is recommended to use `sgx_is_within_enclave` with a size of 1.

Requirements

Header	`sgx_trts.h`
Library	`sgx_trts.lib` or `sgx_trts_sim.lib` (simulation)

sgx_is_outside_enclave

The `sgx_is_outside_enclave` function checks that the buffer located at the pointer `addr` with its length of `size` is an address that is strictly outside enclave address space.

Syntax

```
int sgx_is_outside_enclave (
    const void *addr,
    size_t size
);
```

Parameters

addr [in]

The start address of the buffer.

size [in]

The size of the buffer.

Return value

1

The buffer is strictly outside the enclave address space.

0

The whole buffer or part of the buffer is not outside the enclave, or the buffer is wrapped around.

Description

`sgx_is_outside_enclave` simply compares the start and end address of the buffer with the enclave address space. It does not check the property of the address.

Requirements

Header	`sgx_trts.h`
Library	`sgx_trts.lib` or `sgx_trts_sim.lib` (simulation)

sgx_read_rand

The `sgx_read_rand` function is used to generate a random number inside the enclave.

Syntax

```
sgx_status_t sgx_read_rand(
    unsigned char *rand,
    size_t length_in_bytes
);
```

Parameters

rand [out]

A pointer to the buffer that receives the random number. The pointer cannot be NULL and the rand buffer must be within the enclave.

length_in_bytes [in]

The length of the buffer (in bytes).

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Invalid input parameters detected.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs during the valid random number generation process.

Description

The `sgx_read_rand` function is provided to replace the C standard pseudo-random sequence generation functions inside the enclave, since these standard functions are not supported in the enclave, such as `rand`, `srand`, etc. For HW mode, the function generates a real-random sequence; while for simulation mode, the function generates a pseudo-random sequence.

Requirements

Header	`sgx_trts.h`
Library	`sgx_trts.lib` or `sgx_trts_sim.lib` (simulation)

sgx_register_exception_handler

sgx_register_exception_handler allows developers to register an exception handler, and specify whether to prepend (when is_first_handler is equal to 1) or append the handler to the handler chain.

Syntax

```
void* sgx_register_exception_handler(
    int is_first_handler,
    sgx_exception_handler_t exception_handler
);
```

Parameters

is_first_handler [in]

Report the order in which the handler should be called. If the parameter is nonzero, the handler is the first handler to be called. If the parameter is zero, the handler is the last handler to be called.

exception_handler [in]

The exception handler to be called

Return value

Non-zero

Indicates the exception handler is registered successfully. The return value is an open handle to the custom exception handler.

NULL

The exception handler was not registered.

Description

The Intel® SGX Evaluation SDK supports exception handling with a Vector Exception Handling like API. You can write your own code to handle a limited set of hardware exceptions. For example, a CPUID instruction inside an enclave will effectively result in a #UD fault (Invalid Opcode Exception). ISV enclave code can have an exception handler to prevent the enclave from being trapped into an exception condition. See Custom Exception Handling for more details.

Calling sgx_register_exception_handler allows you to register an exception handler, and specify whether to prepend (when is_first_handler is equal to 1) or append the handler to the handler chain.

NOTE:

Custom exception handling is only supported in hardware mode. Although the exception handlers can be registered in simulation mode, the exceptions cannot be caught and handled within the enclave.

Requirements

Header	sgx_trts_exception.h
Library	sgx_trts.lib or sgx_trts_sim.lib (simulation)

sgx_unregister_exception_handler

sgx_unregister_exception_handler is used to unregister a custom exception handler.

Syntax

```
int sgx_unregister_exception_handler(
    void* handler
);
```

Parameters

handler [in]

A handle to the custom exception handler previously registered using the sgx_register_exception_handler function.

Return value

Non-zero

The custom exception handler is unregistered successfully.

0

The exception handler was not unregistered (not a valid pointer, handler not found).

Description

The Intel® SGX SDK supports exception handling with a Vector Exception Handling like API. An enclave developer can write their own code to handle a limited set of hardware exceptions. See Custom Exception Handling for more details.

Calling sgx_unregister_exception_handler allows developers to unregister an exception handler that was registered earlier.

Requirements

Header	sgx_trts_exception.h
Library	sgx_trts.lib or sgx_trts_sim.lib (simulation)

IsDebuggerPresent

Determines whether the calling process is being debugged by Intel® SGX debugger.

Syntax

```
boolIsDebuggerPresent(
    void
);
```

Parameters

None.

Return value

True

If the current process is running in the context of Intel® SGX debugger.

False

If the current process is not running in the context of Intel® SGX debugger.

Requirements

Header	`sgx_debug.h`
Library	`sgx_trts.lib` or `sgx_trts_sim.lib` (simulation)

OutputDebugString

Sends a string to the Intel® SGX debugger for display.

Syntax

```
void OutputDebugStringA(W)(
     char*(wchar_t*) output_string
);
```

Parameters

lpOutputString [in, optional]

The null-terminated string to be displayed.

Return value

This function does not return a value.

Requirements

Header	`sgx_debug.h`
Library	`sgx_trts.lib` or `sgx_trts_sim.lib` (simulation), `sgx_tstdcxx.lib`

DebugBreak

Causes a breakpoint exception to occur in the current process. This allows the calling thread to signal Intel® SGX debugger to handle the exception.

NOTE:

If an enclave calls this function after the Intel® SGX debugger detaches from the enclave, the application will crash.

Syntax

```
void DebugBreak(
     void
);
```

Parameters

None

Return value

This function does not return a value.

Requirements

Header	sgx_debug.h
Library	sgx_trts.lib or sgx_trts_sim.lib (simulation)

sgx_spin_lock

The sgx_spin_lock function acquires a spin lock within the enclave.

Syntax

```
uint32_t sgx_spin_lock(
     sgx_spinlock_t * lock
);
```

Parameters

lock [in]

The trusted spin lock object to be acquired.

Return value

0

This function always returns zero after the lock is acquired.

Description

sgx_spin_lock modifies the value of the spin lock by using compiler atomic operations. If the lock is not available to be acquired, the thread will always wait on the lock until it can be acquired successfully.

Requirements

Header	sgx_spinlock.h
Library	sgx_tstdc.lib

sgx_spin_unlock

The sgx_spin_unlock function releases a spin lock within the enclave.

Syntax

```
uint32_t sgx_spin_unlock(
     sgx_spinlock_t * lock
);
```

Parameters

lock [in]

The trusted spin lock object to be released.

Return value

0

This function always returns zero after the lock is released.

Description

sgx_spin_unlock resets the value of the spin lock, regardless of its current state. This function simply assigns a value of zero to the lock, which indicates the lock is released.

Requirements

Header	sgx_spinlock.h
Library	sgx_tstdc.lib

sgx_thread_mutex_init

The sgx_thread_mutex_init function initializes a trusted mutex object within the enclave.

Syntax

```
int sgx_thread_mutex_init(
    sgx_thread_mutex_t * mutex,
    const sgx_thread_mutexattr_t * unused
);
```

Parameters

mutex [in]

The trusted mutex object to be initialized.

unused [in]

Unused parameter reserved for future user defined mutex attributes. [NOT USED]

Return value

0

The mutex is initialized successfully.

EINVAL

The trusted mutex object is invalid. It is either NULL or located outside of enclave memory.

Description

When a thread creates a mutex within an enclave, sgx_thread_mutx_init simply initializes the various fields of the mutex object to indicate that the mutex is available. sgx_thread_mutex_init creates a non-recursive mutex. The results of using a mutex in a lock or unlock operation before it has been fully initialized (for example, the function call to sgx_thread_mutex_init returns) are undefined. To avoid race conditions in the initialization of a trusted mutex, it is recommended statically initializing the mutex with the macro SGX_THREAD_MUTEX_INITIALIZER, SGX_THREAD_NON_RECURSIVE_MUTEX_INITIALIZER ,of, or SGX_THREAD_RECURSIVE_MUTEX_INITIALIZER instead.

Requirements

Header	sgx_thread.h sgx_tstdc.edl
Library	sgx_tstdc.lib

sgx_thread_mutex_destroy

The sgx_thread_mutex_destroy function destroys a trusted mutex object within an enclave.

Syntax

```
int sgx_thread_mutex_destroy(
     sgx_thread_mutex_t * mutex
);
```

Parameters

mutex [in]

The trusted mutex object to be destroyed.

Return value

0

The mutex is destroyed successfully.

EINVAL

The trusted mutex object is invalid. It is either NULL or located outside of enclave memory.

EBUSY

The mutex is locked by another thread or has pending threads to acquire the mutex.

Description

`sgx_thread_mutex_destroy` resets the mutex, which brings it to its initial status. In this process, certain fields are checked to prevent releasing a mutex that is still owned by a thread or on which threads are still waiting.

NOTE:

Locking or unlocking a mutex after it has been destroyed results in undefined behavior. After a mutex is destroyed, it must be re-created before it can be used again.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_mutex_lock

The `sgx_thread_mutex_lock` function locks a trusted mutex object within an enclave.

Syntax

```
int sgx_thread_mutex_lock(
     sgx_thread_mutex_t * mutex
);
```

Parameters

mutex [in]

The trusted mutex object to be locked.

Return value

0

The mutex is locked successfully.

EINVAL

The trusted mutex object is invalid.

Description

To acquire a mutex, a thread first needs to acquire the corresponding spin lock. After the spin lock is acquired, the thread checks whether the mutex is available. If the queue is empty or the thread is at the head of the queue the thread will now become the owner of the mutex. To confirm its ownership, the thread updates the refcount and owner fields. If the mutex is not available, the thread searches the queue. If the thread is already in the queue, but not at the head, it means that the thread has previously tried to lock the mutex, but it did not succeed and had to wait outside the enclave and it has been awakened unexpectedly. When this happens, the thread makes an OCALL and simply goes back to sleep. If the thread is trying to lock the mutex for the first time, it will update the waiting queue and make an OCALL to get suspended. Note that threads release the spin lock after acquiring the mutex or before leaving the enclave.

Requirements

Header	`sgx_thread.h sgx_tsrdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_mutex_trylock

The `sgx_thread_mutex_trylock` function tries to lock a trusted mutex object within an enclave.

Syntax

```
int sgx_thread_mutex_trylock(
    sgx_thread_mutex_t * mutex
);
```

Parameters

mutex [in]

The trusted mutex object to be try-locked.

Return value

0

The mutex is locked successfully.

EINVAL

The trusted mutex object is invalid.

EBUSY

The mutex is locked by another thread or has pending threads to acquire the mutex.

Description

A thread may check the status of the mutex, which implies acquiring the spin lock and verifying that the mutex is available and that the queue is empty or the thread is at the head of the queue. When this happens, the thread acquires the mutex, releases the spin lock and returns 0. Otherwise, the thread releases the spin lock and returns EINVAL/EBUSY. The thread is not suspended in this case.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_mutex_unlock

The `sgx_thread_mutex_unlock` function unlocks a trusted mutex object within an enclave.

Syntax

```
int sgx_thread_mutex_unlock(
    sgx_thread_mutex_t * mutex
);
```

Parameters

mutex [in]

The trusted mutex object to be unlocked.

Return value

0

The mutex is unlocked successfully.

EINVAL

The trusted mutex object is invalid or it is not locked by any thread.

EPERM

The mutex is locked by another thread.

Description

Before a thread releases a mutex, it has to verify it is the owner of the mutex. If that is the case, the thread decreases the refcount by 1 and then may either continue normal execution or wakeup the first thread in the queue. Note that to ensure the state of the mutex remains consistent, the thread that is awakened by the thread releasing the mutex will then try to acquire the mutex almost as in the initial call to the `sgx_thread_mutex_lock` routine.

Requirements

Header	`sgx_thread.h sgxtstdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_cond_init

The `sgx_thread_cond_init` function initializes a trusted condition variable within the enclave.

Syntax

```
int sgx_thread_cond_init(
    sgx_thread_cond_t * cond,
    const sgx_thread_condattr_t * unused
);
```

Parameters

cond [in]

The trusted condition variable.

attr [in]

Unused parameter reserved for future user defined condition variable attributes. [NOT USED]

Return value

0

The condition variable is initialized successfully.

EINVAL

The trusted condition variable is invalid. It is either NULL or located outside enclave memory.

Description:

When a thread creates a condition variable within an enclave, it simply initializes the various fields of the object to indicate that the condition variable is available. The results of using a condition variable in a wait, signal or broadcast operation before it has been fully initialized (for example, the function call to `sgx_thread_cond_init` returns) are undefined. To avoid race conditions in the initialization of a condition variable, it is recommended statically initializing the condition variable with the macro `SGX_THREAD_COND_INITIALIZER`.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_cond_destroy

The `sgx_thread_cond_destroy` function destroys a trusted condition variable within an enclave.

```
Syntax
int sgx_thread_cond_destroy(
    sgx_thread_cond_t * cond
);
```

Parameters

cond [in]

The trusted condition variable to be destroyed.

Return value

0

The condition variable is destroyed successfully.

EINVAL

The trusted condition variable is invalid. It is either NULL or located outside enclave memory.

EBUSY

The condition variable has pending threads waiting on it.

Description

The procedure first confirms that there are no threads waiting on the condition variable before it is destroyed. The destroy operation acquires the spin lock at the beginning of the operation to prevent other threads from signaling to or waiting on the condition variable.

NOTE

Acquiring or releasing a condition variable after it has been destroyed results in undefined behavior. After a condition variable is destroyed, it must be re-created before it can be used again.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_cond_wait

The `sgx_thread_cond_wait` function waits on a condition variable within an enclave.

Syntax

```
int sgx_thread_cond_wait(
     sgx_thread_cond_t * cond,
     sgx_thread_mutex_t * mutex
);
```

Parameters

cond [in]

The trusted condition variable to be waited on.

mutex [in]

The trusted mutex object that will be unlocked when the thread is blocked in the condition variable.

Return value

0

The thread waiting on the condition variable is signaled by other thread (without errors).

EINVAL

The trusted condition variable or mutex object is invalid or the mutex is not locked.

EPERM

The trusted mutex is locked by another thread.

Description:

A condition variable is always used in conjunction with a mutex. To wait on a condition variable, a thread first needs to acquire the condition variable spin lock. After the spin lock is acquired, the thread updates the condition variable waiting queue. To avoid the lost wake-up signal problem, the condition variable spin lock is released after the mutex. This order ensures the function atomically releases the mutex and causes the calling thread to block on the condition variable, with respect to other threads accessing the mutex and the condition variable. After releasing the condition variable spin lock, the thread makes an OCALL to get suspended. When the thread is awakened, it acquires the condition variable spin lock. The thread then searches the condition variable queue. If the thread is in the queue, it means that the thread was already waiting on the condition variable outside the enclave, and it has been awakened unexpectedly. When this happens, the thread releases the condition variable spin lock, makes an OCALL and simply goes back to sleep. Otherwise, another thread has signaled or broadcasted the condition variable and this thread may proceed. Before returning, the thread releases the condition variable spin lock and acquires the mutex, ensuring that upon returning from the function call the thread still owns the mutex.

NOTE

Threads check whether they are in the queue to make the Intel SGX condition variable robust against attacks to the untrusted event.

A thread may have to do up to two OCALLs throughout the `sgx_thread_cond_wait` function call.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_cond_signal

The `sgx_thread_cond_signal` function wakes a pending thread waiting on the condition variable.

Syntax

```
int sgx_thread_cond_signal(
    sgx_thread_cond_t * cond
);
```

Parameters

cond [in]

The trusted condition variable to be signaled.

Return value

0

One pending thread is signaled.

EINVAL

The trusted condition variable is invalid.

Description

To signal a condition variable, a thread starts acquiring the condition variable spin-lock. Then it inspects the status of the condition variable queue. If the queue is empty it means that there are not any threads waiting on the condition variable. When that happens, the thread releases the condition variable and returns. However, if the queue is not empty, the thread removes the first thread waiting in the queue. The thread then makes an OCALL to wake up the thread that is suspended outside the enclave, but first the thread releases the condition variable spin-lock. Upon returning from the OCALL, the thread continues normal execution.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_cond_broadcast

The `sgx_thread_cond_broadcast` function wakes all pending threads waiting on the condition variable.

Syntax

```
int sgx_thread_cond_broadcast(
     sgx_thread_cond_t * cond
);
```

Parameters

cond [in]

The trusted condition variable to be broadcasted.

Return value

0

All pending threads have been broadcasted.

EINVAL

The trusted condition variable is invalid.

ENOMEM

Internal memory allocation failed.

Description

Broadcast and signal operations on a condition variable are analogous. The only difference is that during a broadcast operation, the thread removes all the threads waiting on the condition variable queue and wakes up all the threads suspended outside the enclave in a single OCALL.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_thread_self

The `sgx_thread_self` function returns the unique thread identification.

Syntax

```
sgx_thread_t sgx_thread_self(
     void
);
```

Return value

The return value cannot be NULL and is always valid as long as it is invoked by a thread inside the enclave.

Description

The function is a simple wrap of `get_thread_data()` provided in the tRTS, which provides a trusted thread unique identifier.

NOTE:

This identifier does not change throughout the life of an enclave.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_cpuid

The `sgx_cpuid` function performs the equivalent of a cpuid() function call or intrinisic which executes the CPUID instruction to query the host processor for the information about supported features.

NOTE:

This function performs an OCALL to execute the CPUID instruction.

Syntax

```
sgx_status_t sgx_cpuid(
     int cpuinfo[4],
     int leaf
);
```

Parameters

cpuinfo [in, out]

The information returned in an array of four integers. This array must be located within the enclave.

leaf [in]

The leaf specified for retrieved CPU info.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates the parameter cpuinfo is invalid, which would be NULL or outside the enclave.

Description

This function provides the equivalent of the cpuid() function or intrinsic. The function executes the CPUID instruction for the given leaf (input). The CPUID instruction provides processor feature and type information that is returned in cpuinfo, an array of 4 integers to specify the values of EAX, EBX, ECX and EDX registers. `sgx_cpuid` performs an OCALL by invoking oc_cpuidex to get the info from untrusted side because the CPUID instruction is an illegal instruction in the enclave domain.

For additional details, see Intel® 64 and IA-32 Architectures Software Developer's Manual for the description on the CPUID instruction and its individual leafs. (Leaf corresponds to EAX in the PRM description).

NOTE

1. As the CPUID instruction is executed by an OCALL, the results should not be trusted. Code should verify the results and perform a threat evaluation to determine the impact on trusted code if the results were spoofed.
2. The implementation of this function performs an OCALL and therefore, this function will not have the same serializing or fencing behavior of executing a CPUID instruction in an untrusted domain code flow.

Requirements

Header	`sgx_cpuid.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_cpuidex

The `sgx_cpuidex` function performs the equivalent of a `cpuid_ex()` function call or intrinisic which executes the CPUID instruction to query the host processor for the information about supported features.

NOTE:

This function performs an OCALL to execute the CPUID instruction.

Syntax

```
sgx_status_t sgx_cpuidex(
    int cpuinfo[4],
    int leaf,
    int subleaf
);
```

Parameters

cpuinfo [in, out]

The information returned in an array of four integers. The array must be located within the enclave.

leaf[in]

The leaf specified for retrieved CPU info.

subleaf[in]

The sub-leaf specified for retrieved CPU info.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates the parameter cpuinfo is invalid, which would be NULL or outside the enclave.

Description

This function provides the equivalent of the `cpuid()` function or intrinsic. The function executes the CPUID instruction for the given leaf (input). The CPUID instruction provides processor feature and type information returned in cpuinfo, an array of 4 integers to specify the values of EAX, EBX, ECX and EDX registers. sgx_cpuid performs an OCALL by invoking oc_cpuidex to get the info from untrusted side because the CPUID instruction is an illegal instruction in the enclave domain.

For additional details, see Intel® 64 and IA-32 Architectures Software Developer's Manual for the description on the CPUID instruction and its individual leafs. (Leaf corresponds to EAX in the PRM description).

NOTE

1. As the CPUID instruction is executed by an OCALL, the results should not be trusted. Code should verify the results and perform a threat evaluation to determine the impact on trusted code if the results were spoofed.
2. The implementation of this function performs an OCALL and therefore, this function will not have the same serializing or fencing behavior of executing a CPUID instruction in an untrusted domain code flow.

Requirements

Header	`sgx_cpuid.h sgx_tstdc.edl`
Library	`sgx_tstdc.lib`

sgx_get_key

The `sgx_get_key` function generates a 128-bit secret key using the input information. This function is a wrapper for the SGX EGETKEY instruction.

Syntax

```
sgx_status_t sgx_get_key(
    const sgx_key_request_t *key_request,
    sgx_key_128bit_t *key
);
```

Parameters

key_request [in]

A pointer to a sgx_key_request_t object used for selecting the appropriate key and any additional parameters required in the derivation of that key. The pointer cannot be NULL and must be located within the enclave. See details on the sgx_key_request_t to understand initializing this structure before calling this function.

key [out]

A pointer to the buffer that receives the cryptographic key output. The pointer cannot be NULL and must be located within enclave memory.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates an error if the parameters do not meet any of the following conditions:

`key_request` buffer must be non-NULL and located within the enclave.

key buffer must be non-NULL and located within the enclave.

`key_request->key_policy` should not have any reserved bits set.

SGX_ERROR_OUT_OF_MEMORY

Indicates an error that the enclave is out of memory.

SGX_ERROR_INVALID_ATTRIBUTE

Indicates the `key_request` requests a key for a `KEYNAME` which the enclave is not authorized.

SGX_ERROR_INVALID_CPUSVN

Indicates `key_request->cpu_svn` is beyond platform CPUSVN value

SGX_ERROR_INVALID_ISVSVN

Indicates `key_request->isv_svn` is greater than the enclave’s ISVSVN

SGX_ERROR_INVALID_KEYNAME

Indicates `key_request->key_name` is an unsupported value

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs during the key generation process.

Description

The `sgx_get_key` function generates a 128-bit secret key from the processor specific key hierarchy with the `key_request` information. If the function fails with an error code, the key buffer will be filled with random numbers. The `key_request` structure needs to be initialized properly to obtain the requested key type. See sgx_key_request_t for structure details.

Requirements

Header	`sgx_utils.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_create_report

The `sgx_create_report` function tries to use the information of the target enclave and other information to create a cryptographic report of the enclave. This function is a wrapper for the `SGX EREPORT` instruction.

Syntax

```
sgx_status_t sgx_create_report(
     const sgx_target_info_t *target_info,
     const sgx_report_data_t *report_data,
     sgx_report_t *report
);
```

Parameters

target_info [in]

A pointer to the sgx_target_info_t object that contains the information of the target enclave. The pointer is allowed to be NULL. If it is not NULL, the `target_info` buffer must be within the enclave. See documentation on sgx_target_info_t for structure details.

report_data [in]

A pointer to the sgx_report_data_t object which contains a set of data used for communication between the enclaves. This pointer is allowed to be NULL. If it is not NULL, the `report_data` buffer must be within the enclave. See sgx_report_data_t for structure details.

report [out]

A pointer to the buffer that receives the cryptographic report of the enclave. The pointer cannot be NULL and the report buffer must be within the enclave. See sgx_report_t for structure details.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

An error is reported if any of the parameters are non-NULL pointers but the memory is not within the enclave or the reserved fields of the data structure are not set to zero.

SGX_ERROR_OUT_OF_MEMORY

Indicates that the enclave is out of memory.

Description

The function `sgx_create_report` is used to create a cryptographic report that describes the contents of the enclave. The cryptographic report can be used by other enclaves to determine that the enclave is running on the same platform. This function is a wrapper for the `SGX EREPORT` instruction.

Requirements

Header	`sgx_utils.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_verify_report

The `sgx_verify_report` function provides software verification for the report which is expected to be generated by the sgx_create_report function.

Syntax

```
sgx_status_t sgx_verify_report(
     const sgx_report_t * report
);
```

Parameters

report[in]

A pointer to an sgx_report_t object that contains the cryptographic report to be verified. The pointer cannot be NULL and the report buffer must be within the enclave.

Return value

SGX_SUCCESS

Verification success.

SGX_ERROR_INVALID_PARAMETER

The report object is invalid.

SGX_ERROR_MAC_MISMATCH

Indicates report verification error.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs during the report verification process.

Description

The `sgx_verify_report` performs a cryptographic CMAC function of the input sgx_report_data_t object in the report using the report key. Then the function compares the input report MAC value with the calculated MAC value to determine whether the report is valid or not.

Requirements

Header	`sgx_utils.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_calc_sealed_data_size

The `sgx_calc_sealed_data_size` function is a helper function for the seal library which should be used to determine how much memory to allocate for the sgx_sealed_data_t structure.

Syntax

```
uint32_t sgx_calc_sealed_data_size(
    const uint32_t add_mac_txt_size,
    const uint32_t txt_encrypt_size
);
```

Parameters

add_mac_txt_size [in]

Length of the optional additional data stream in bytes. The additional data will not be encrypted, but will be part of the MAC calculation.

txt_encrypt_size [in]

Length of the data stream to be encrypted in bytes. This data will also be part of the MAC calculation.

Return value

If the function succeeds, the return value is the minimum number of bytes that need to be allocated for the sgx_sealed_data_t structure. If the function fails, the return value is 0xFFFFFFFF. It is recommended that you check the return value before use the function to allocate memory.

Description

The function calculates the number of bytes to allocate for the sgx_sealed_data_t structure. The calculation includes the fixed portions of the structure as well as the two input data streams: encrypted text and optional additional MAC text.

Requirements

Header	sgx_tseal.h
Library	sgx_tservice.lib or sgx_tservice_sim.lib (simulation)

sgx_get_add_mac_txt_len

The sgx_get_add_mac_txt_len function is a helper function for the seal library which should be used to determine how much memory to allocate for the additional_MAC_text buffer output from the sgx_unseal_data function.

Syntax

```
uint32_t sgx_get_add_mac_txt_len(
    const sgx_sealed_data_t *p_sealed_data
);
```

Parameters

p_sealed_data [in]

Pointer to the sealed data structure which was populated by the sgx_seal_data function.

Return value

If the function succeeds, the number of bytes in the optional additional MAC data buffer is returned. If this function fails, the return value is 0xFFFFFFFF. It is recommended that you check the return value before use the function to allocate memory.

Description

The function calculates the minimum number of bytes to allocate for the output MAC data buffer returned by the sgx_unseal_data function.

Requirements

Header	sgx_tseal.h
Library	sgx_tservice.lib or sgx_tservice_sim.lib (simulation)

sgx_get_encrypt_txt_len

The sgx_get_encrypt_txt_len function is a helper function for the seal library which should be used to calculate the minimum number of bytes to allocate for decrypted data returned by the sgx_unseal_data function.

Syntax

```
uint32_t sgx_get_encrypt_txt_len(
    const sgx_sealed_data_t *p_sealed_data
```

```
);
```

Parameters

p_sealed_data [in]

Pointer to the sealed data structure which was populated during by the sgx_seal_data function.

Return value

If the function succeeds, the number of bytes in the encrypted data buffer is returned. Othewise, the return value is `0xFFFFFFFF`. It is recommended that you check the return value before use the function to allocate memory.

Description

The function calculates the minimum number of bytes to allocate for decrypted data returned by the sgx_unseal_data function.

Requirements

Header	`sgx_tseal.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_seal_data

This function is used to AES-GCM encrypt the input data. Two input data sets are provided: one is the data to be encrypted; the second is optional additional data that will not be encrypted but will be part of the GCM MAC calculation which also covers the data to be encrypted.

Syntax

```
sgx_status_t sgx_seal_data(
    const uint32_t additional_MACtext_length,
    const uint8_t * p_additional_MACtext,
    const uint32_t text2encrypt_length,
    const uint8_t * p_text2encrypt,
    const uint32_t sealed_data_size,
    sgx_sealed_data_t * p_sealed_data
);
```

Parameters

additional_MACtext_length [in]

Length of the additional Message Authentication Code (MAC) data in bytes. The additional data is optional and thus the length can be zero if no data is provided.

p_addtional_MACtext [in]

Pointer to the additional Message Authentication Code (MAC) data. This additional data is optional and no data is necessary (NULL pointer can be passed, but `additional_MACtext_length` must be zero in this case).

NOTE:

This data will not be encrypted. This data can be within or outside the enclave, but cannot cross the enclave boundary.

---

text2encrypt_length [in]

Length of the data stream to be encrypted in bytes. Must be non-zero.

p_text2encrypt [in]

Pointer to the data stream to be encrypted. Must not be NULL. Must be within the enclave.

sealed_data_size [in]

Number of bytes allocated for the sgx_sealed_data_t structure. The calling code should utilize helper function `sgx_calc_sealed_data_size` to determine the required buffer size.

p_sealed_data [out]

Pointer to the buffer to store the sealed data.

NOTE:

The calling code must allocate the memory for this buffer and should utilize helper function `sgx_calc_sealed_data_size` to determine the required buffer size. The sealed data must be within the enclave.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates an error if the parameters do not meet any of the following conditions:

- If `additional_mactext_length` is non-zero, `p_additional_mactext` cannot be NULL.
- `p_additional_mactext` buffer can be within or outside the enclave, but cannot cross the enclave boundary.
- `p_text2encrypt` must be non-zero.
- `p_text2encrypt` buffer must be within the enclave.
- `sealed_data_size` must be equal to the required buffer size, which is calculated by the function `sgx_calc_sealed_data_size`.
- `p_sealed_data` buffer must be within the enclave.
- Input buffers cannot cross an enclave boundary.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

Indicates a crypto library failure or the RDRAND instruction fails to generate a random number.

Description

The `sgx_seal_data` function retrieves a key unique to the enclave and uses that key to encrypt the input data buffer. This function can be utilized to preserve secret data after the enclave is destroyed. The sealed data blob can be unsealed on future instantiations of the enclave.

The additional data buffer will not be encrypted but will be part of the MAC calculation that covers the encrypted data as well. This data may include information about the application, version, data, etc which can be utilized to identify the sealed data blob since it will remain plain text

Use `sgx_calc_sealed_data_size` to calculate the number of bytes to allocate for the sgx_sealed_data_t structure. The input sealed data buffer and text2encrypt buffers must be allocated within the enclave.

Requirements

Header	`sgx_tseal.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_seal_data_ex

This function is used to AES-GCM encrypt the input data. Two input data sets are provided: one is the data to be encrypted; the second is optional additional data that will not be encrypted but will be part of the GCM MAC calculation which also covers the data to be encrypted. This is the expert mode version of function `sgx_seal_data`.

Syntax

```
sgx_status_t sgx_seal_data_ex(
    const uint16_t key_policy,
    const sgx_attributes_t attribute_mask,
    const sgx_misc_select_t misc_mask,
    const uint32_t additional_MACtext_length,
    const uint8_t * p_additional_MACtext,
    const uint32_t text2encrypt_length,
    const uint8_t * p_text2encrypt,
    const uint32_t sealed_data_size,
    sgx_sealed_data_t * p_sealed_data
);
```

Parameters

key_policy [in]

Specifies the policy to use in the key derivation. Function `sgx_seal_data` uses the MRSIGNER policy.

Key policy name Value Description

Key policy name	Value	Description
`KEYPOLICY_MRENCLAVE`	`0x0001`	Derive key using the enclave's ENCLAVE measurement register
`KEYPOLICY_MRSIGNER`	`0x0002`	Derive key using the enclave's SIGNER measurement register

attribute_mask [in]

Identifies which platform/enclave attributes to use in the key derivation. See the definition of sgx_attributes_t to determine which attributes will be checked. Function sgx_seal_data uses `flags=0xfffffffffffffff3, xfrm=0`.

misc_mask [in]

The misc mask bits for the enclave. Reserved for future function extension.

additional_MACtext_length [in]

Length of the additional data to be MAC'ed in bytes. The additional data is optional and thus the length can be zero if no data is provided.

p_addtional_MACtext [in]

Pointer to the additional data to be MAC'ed of variable length. This additional data is optional and no data is necessary (NULL pointer can be passed, but `additional_MACtext_length` must be zero in this case).

NOTE:

This data will not be encrypted. This data can be within or outside the enclave, but cannot cross the enclave boundary.

---

text2encrypt_length [in]

Length of the data stream to be encrypted in bytes. Must be non-zero.

p_text2encrypt [in]

Pointer to the data stream to be encrypted of variable length. Must not be NULL. Must be within the enclave.

sealed_data_size [in]

Number of bytes allocated for `sealed_data_t` structure. The calling code should utilize helper function `sgx_calc_sealed_data_size` to determine the required buffer size.

p_sealed_data [out]

Pointer to the buffer that is populated by this function.

NOTE:

The calling code must allocate the memory for this buffer and should utilize helper function `sgx_calc_sealed_data_size` to determine the required buffer size. The sealed data must be within the enclave.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates an error if the parameters do not meet any of the following conditions:

- `If additional_mactext_length` is non-zero, `p_additional_mactext` cannot be NULL.
- `p_additional_mactext` buffer can be within or outside the enclave, but cannot cross the enclave boundary.
- `p_text2encrypt` must be non-zero.
- `p_text2encrypt` buffer must be within the enclave.

- sealed_data_size must be equal to the required buffer size, which is calculated by the function sgx_calc_sealed_data_size.
- p_sealed_data buffer must be within the enclave.
- Input buffers cannot cross an enclave boundary.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

Indicates crypto library failure or the RDRAND instruction fails to generate a random number.

Description

The sgx_seal_data_ex is an extended version of sgx_seal_data. It provides parameters for you to identify how to derive the sealing key (key policy and attributes_mask). Typical callers of the seal library should be able to use sgx_seal_data and the default values provided for key_policy (MR_SIGNER) and an attribute mask which includes the RESERVED, INITED and DEBUG bits. Users of this function should have a clear understanding of the impact on using a policy and/or attribute_mask that is different from that in sgx_seal_data.

Requirement

Header	sgx_tseal.h
Library	sgx_tservice.lib or sgx_tservice_sim.lib (simulation)

sgx_unseal_data

This function is used to AES-GCM decrypt the input sealed data structure. Two output data sets result: one is the decrypted data; the second is the optional additional data that was part of the GCM MAC calculation but was not encrypted. This function provides the converse of sgx_seal_data and sgx_seal_data_ex.

Syntax

```
sgx_status_t sgx_unseal_data(
     const sgx_sealed_data_t * p_sealed_data,
     uint8_t * p_additional_MACtext,
     uint32_t * p_additional_MACtext_length,
     uint8_t * p_decrypted_text,
     uint32_t * p_decrypted_text_length
);
```

Parameters

p_sealed_data [in]

Pointer to the sealed data buffer to be AES-GCM decrypted. Must be within the enclave.

p_addtional_MACtext [out]

Pointer to the additional data part of the MAC calculation. This additional data is optional and no data is necessary. The calling code should call helper function sgx_get_mac_add_text_len to determine the required buffer size to allocate. (NULL pointer can be passed, if additional_MACtext_length is zero).

p_additional_MACtext_length [in, out]

Pointer to the length of the additional MAC data buffer in bytes. The calling code should call helper function sgx_get_mac_add_text_len to determine the minimum required buffer size. The sgx_unseal_data function returns the actual length of decrypted addition data stream.

p_decrypted_text [out]

Pointer to the decrypted data buffer which needs to be allocated by the calling code. Use sgx_get_encrypt_txt_len to calculate the minimum number of bytes to allocate for the p_decrypted_text buffer. Must be within the enclave.

p_decrypted_text_length [in, out]

Pointer to the length of the decrypted data buffer in byte. The buffer length of p_decrypted_text must be specified in p_decrypted_text_length as input. The sgx_unseal_data function returns the actual length of decrypted addition data stream. Use sgx_get_encrypt_txt_len to calculate the number of bytes to allocate for the p_decrypted_text buffer. Must be within the enclave.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates an error if the parameters do not meet any of the following conditions:

- If additional_mactext_length is non-zero, p_additional_mactext cannot be NULL.
- p_additional_mactext buffer can be within or outside the enclave, but cannot across the enclave boundary.
- p_decrypted_text and p_decrypted_text_length must be within the enclave.
- p_decrypted_text and p_addtitional_MACtext buffer must be big enough to receive the decrypted data.
- p_sealed_data buffer must be within the enclave.
- Input buffers cannot cross an enclave boundary.

SGX_ERROR_INVALID_CPUSVN

The CPUSVN in the sealed data blob is beyond the CPUSVN value of the platform.

SGX_ERROR_INVALID_ISVSVN

The ISVSVN in the sealed data blob is greater than the ISVSVN value of the enclave.

SGX_ERROR_MAC_MISMATCH

The tag verification failed during unsealing. The error may be caused by a platform update, software update, or sealed data blob corruption. This error is also reported if other corruption of the sealed data structure is detected.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

Indicates a cryptography library failure.

Description

The sgx_unseal_data function AES-GCM decrypts the sealed data so that the enclave data can be restored. This function can be utilized to restore secret data that was preserved after an earlier instantiation of this enclave saved this data.

The calling code needs to allocate the additional data buffer and the decrypted data buffer. To determine the minimum memory to allocate for these buffers, helper functions `sgx_get_mac_add_text_len` and `sgx_get_encrypt_txt_len` are provided. The decrypted text buffer must be allocated within the enclave.

Requirements

Header	`sgx_tseal.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_sha256_msg

The `sgx_sha256_msg` function performs a standard SHA256 hash over the input data buffer.

Syntax

```
sgx_status_t sgx_sha256_msg(
    const uint8_t *p_src,
    uint32_t src_len,
    sgx_sha256_hash_t *p_hash
);
```

Parameters

p_src [in]

A pointer to the input data stream to be hashed. A zero length input buffer is supported, but the pointer must be non-NULL.

src_len [in]

Specifies the length on the input data stream to be hashed. A zero length input buffer is supported.

p_hash [out]

A pointer to the output 256bit hash resulting from the SHA256 calculation. This pointer must be non-NULL and the caller allocates memory for this buffer.

Return value

SGX_SUCCESS

The SHA256 hash function is performed successfully.

SGX_ERROR_INVALID_PARAMETER

Input pointers are invalid.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The SHA256 hash calculation failed.

Description

The `sgx_sha256_msg` function performs a standard SHA256 hash over the input data buffer. Only a 256-bit version of the SHA hash is supported. (Other sizes, example 512, are not supported in this minimal cryptography library).

The function should be used if the complete input data stream is available. Otherwise, the Init, Update… Update, Final procedure should be used to compute a SHA256 bit hash over multiple input data sets.

A zero-length input data buffer is supported but the pointer must be non-NULL.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_sha256_init

`sgx_sha256_init` returns an allocated and initialized SHA algorithm context state. This should be part of the Init, Update … Update, Final process when the SHA hash is to be performed over multiple datasets. If a complete dataset is available, the recommend call is `sgx_sha256_msg` to perform the hash in a single call.

Syntax

```
sgx_status_t sgx_sha256_init(
    sgx_sha_state_handle_t* p_sha_handle
);
```

Parameters

p_sha_handle [out]

This is a handle to the context state used by the cryptography library to perform an iterative SHA256 hash. The algorithm stores the intermediate results of performing the hash calculation over data sets.

Return value

SGX_SUCCESS

The SHA256 state is allocated and initialized properly.

SGX_ERROR_INVALID_PARAMETER

The pointer `p_sha_handle` is invalid.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The SHA256 state is not initialized properly due to an internal cryptography library failure.

Description

Calling `sgx_sha256_init` is the first set in performing a SHA256 hash over multiple datasets. The caller does not allocate memory for the SHA256 state that this function returns. The state is specific to the implementation of the cryptography library; thus the allocation is performed by the library itself. If the hash over the desired datasets is completed or any error occurs during the hash calculation process, `sgx_sha256_close` should be called to free the state allocated by this algorithm.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_sha256_update

`sgx_sha256_update` performs a SHA256 hash over the input dataset provided. This function supports an iterative calculation of the hash over multiple datasets where the sha_handle contains the intermediate results of the hash calculation over previous datasets.

Syntax

```
sgx_status_t sgx_sha256_update(
    const uint8_t *p_src,
    uint32_t src_len,
    sgx_sha_state_handle_tsha_handle
);
```

Parameters

p_src [in]

A pointer to the input data stream to be hashed. A zero length input buffer is supported, but the pointer must be non-NULL.

src_len [in]

Specifies the length on the input data stream to be hashed. A zero length input buffer is supported.

sha_handle [in]

This is a handle to the context state used by the cryptography library to perform an iterative SHA256 hash. The algorithm stores the intermediate results of performing the hash calculation over multiple data sets.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

The input parameter(s) are NULL.

SGX_ERROR_UNEXPECTED

An internal cryptography library failure occurred while performing the SHA256 hash calculation.

Description

This function should be used as part of a SHA256 calculation over multiple datasets. If a SHA256 hash is needed over a single data set, function `sgx_sha256_msg` should be used instead. Prior to calling this function on the first dataset, the sgx_sha256_init function must be called first to allocate and initialize the SHA256 state structure which will hold intermediate hash results over earlier datasets. The function `sgx_sha256_get_hash` should be used to obtain the hash after the final dataset has been processed by this function.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_sha256_get_hash

`sgx_sha256_get_hash` obtains the SHA256 hash after the final dataset has been processed (by calls to `sgx_sha256_update`).

Syntax

```
sgx_status_t sgx_sha256_get_hash(
    sgx_sha_state_handle_tsha_handle,
    sgx_sha256_hash_t* p_hash
);
```

Parameters

sha_handle [in]

This is a handle to the context state used by the cryptography library to perform an iterative SHA256 hash. The algorithm stores the intermediate results of performing the hash calculation over multiple datasets.

p_hash [out]

This is a pointer to the 256-bit hash that has been calculated. The memory for the hash should be allocated by the calling code.

Return value

SGX_SUCCESS

The hash is obtained successfully.

SGX_ERROR_INVALID_PARAMETER

The pointers are NULL.

SGX_ERROR_UNEXPECTED

The SHA256 state passed in is likely problematic causing an internal cryptography library failure.

Description

This function returns the hash after performing the SHA256 calculation over one or more datasets using the `sgx_sha256_update` function. Memory for the hash should be allocated by the calling function. The handle to SHA256 state used in the `sgx_sha256_update` calls must be passed in as input.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_sha256_close

`sgx_sha256_close` cleans up and deallocates the SHA256 state that was allocated in function `sgx_sha256_init`.

Syntax

```
sgx_status_t sgx_sha256_close(
    sgx_sha_state_handle_tsha_handle
);
```

Parameters

sha_handle [in]

This is a handle to the context state used by the cryptography library to perform an iterative SHA256 hash. The algorithm stores the intermediate results of performing the hash calculation over data sets.

Return value

SGX_SUCCESS

The SHA256 state was deallocated successfully.

SGX_ERROR_INVALID_PARAMETER

The input handle is NULL.

Description

Calling `sgx_sha256_close` is the last step after performing a SHA256 hash over multiple data-sets. The caller uses this function to deallocate memory used to store the SHA256 calculation state.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_rijndael128GCM_encrypt

`sgx_rijndael128GCM_encrypt` performs a Rijndael AES-GCM encryption operation. Only a 128bit key size is supported by this Intel® SGX SDK cryptography library.

Syntax

```
sgx_status_t sgx_rijndael128GCM_encrypt(
     const sgx_aes_gcm_128bit_key_t *p_key,
     const uint8_t *p_src,
     uint32_t src_len,
     uint8_t *p_dst,
     const uint8_t *p_iv,
     uint32_t iv_len,
     const uint8_t *p_aad,
     uint32_t aad_len,
     sgx_aes_gcm_128bit_tag_t *p_out_mac
);
```

Parameters

p_key [in]

A pointer to key to be used in the AES-GCM encryption operation. The size *must* be 128 bits.

p_src [in]

A pointer to the input data stream to be encrypted. Buffer could be NULL if there is AAD text.

src_len [in]

Specifies the length on the input data stream to be encrypted. This could be zero but `p_src` and `p_dst` should be NULL and `aad_len` must be greater than zero.

p_dst [out]

A pointer to the output encrypted data buffer. This buffer should be allocated by the calling code.

p_iv [in]

A pointer to the initialization vector to be used in the AES-GCM calculation. NIST AES-GCM recommended IV size is 96bits (12 bytes).

iv_len [in]

Specifies the length on input initialization vector. The length should be 12 as recommended by NIST.

p_aad [in]

A pointer to an optional additional authentication data buffer which is used in the GCM MAC calculation. The data is this buffer will not be encrypted. The field is optional and could be NULL.

aad_len [in]

Specifies the length of the additional authentication data buffer. This buffer is optional and thus the size can be zero.

p_out_mac [out]

This is the output GCM MAC performed over the input data buffer (data to be encrypted) as well as the additional authentication data (this is optional data). The calling code should allocate this buffer.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

If key, source, destination, MAC, or IV pointer is NULL.

If AAD size is > 0 and the AAD pointer is NULL.

If source size is > 0 and the source pointer or destination pointer are NULL.

If both the AAD size is 0 and the source size is 0.

If IV Length is not equal to 12 (bytes).

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An internal cryptography library failure occurred.

Description

The Galois/Counter Mode (GCM) is a mode of operation of the AES algorithm. GCM [NIST SP 800-38D] uses a variation of the counter mode of operation for encryption. GCM assures authenticity of the confidential data (of up to about 64 GB per invocation) using a universal hash function defined over a binary finite field (the Galois field).

GCM can also provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted. GCM provides stronger authentication assurance than a (non-cryptographic) checksum or error detecting code. In particular, GCM can detect both accidental modifications of the data and intentional, unauthorized modifications.

It is recommended that the source and destination data buffers are allocated within the enclave. The AAD buffer could be allocated within or outside enclave memory. The use of AAD data buffer could be information identifying the encrypted data since it will remain in clear text.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_rijndael128GCM_decrypt

`sgx_rijndael128GCM_decrypt` performs a Rijndael AES-GCM decryption operation. Only a 128bit key size is supported by this Intel® SGX SDK cryptography library.

Syntax

```
sgx_status_t sgx_rijndael128GCM_decrypt(
    const sgx_aes_gcm_128bit_key_t *p_key,
    const uint8_t *p_src,
    uint32_t src_len,
    uint8_t *p_dst,
    const uint8_t *p_iv,
    uint32_t iv_len,
    const uint8_t *p_aad,
    uint32_t aad_len,
    const sgx_aes_gcm_128bit_tag_t *p_in_mac
);
```

Parameters

p_key [in]

A pointer to key to be used in the AES-GCM decryption operation. The size *must* be 128 bits.

p_src [in]

A pointer to the input data stream to be decrypted. Buffer could be NULL if there is AAD text.

src_len [in]

Specifies the length on the input data stream to be decrypted. This could be zero but `p_src` and `p_dst` should be NULL and `aad_len` must be greater than zero.

p_dst [out]

A pointer to the output decrypted data buffer. This buffer should be allocated by the calling code.

p_iv [in]

A pointer to the initialization vector to be used in the AES-GCM calculation. NIST AES-GCM recommended IV size is 96bits (12 bytes).

iv_len [in]

Specifies the length on input initialization vector. The length should be 12 as recommended by NIST.

p_aad [in]

A pointer to an optional additional authentication data buffer which is provided for the GCM MAC calculation when encrypting. The data is this buffer was not encrypted. The field is optional and could be NULL.

aad_len [in]

Specifies the length of the additional authentication data buffer. This buffer is optional and thus the size can be zero.

p_out_mac [out]

This is the GCM MAC that was performed over the input data buffer (data to be encrypted) as well as the additional authentication data (this is optional data) during the encryption process (call to `sgx_rijndael128GCM_encrypt`).

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

If key, source, destination, MAC, or IV pointer is NULL.

If AAD size is > 0 and the AAD pointer is NULL.

If source size is > 0 and the source pointer or destination pointer are NULL.

If both the AAD size is 0 and the source size is 0.

If IV Length is not equal to 12 (bytes).

SGX_ERROR_MAC_MISMATCH

The input MAC does not match the MAC calculated.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An internal cryptography library failure occurred.

Description

The Galois/Counter Mode (GCM) is a mode of operation of the AES algorithm. GCM [NIST SP 800-38D] uses a variation of the counter mode of operation for encryption. GCM assures authenticity of the confidential data (of up to about 64 GB per invocation) using a universal hash function defined over a binary finite field (the Galois field).

GCM can also provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted. GCM provides stronger authentication assurance than a (non-cryptographic) checksum or error detecting code. In particular, GCM can detect both accidental modifications of the data and intentional, unauthorized modifications.

It is recommended that the destination data buffer is allocated within the enclave. The AAD buffer could be allocated within or outside enclave memory.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_rijndael128_cmac_msg

The `sgx_rijndael128_cmac_msg` function performs a standard 128bit CMAC hash over the input data buffer.

Syntax

```
sgx_status_t sgx_rijndael128_cmac_msg(
    const sgx_cmac_128bit_key_t *p_key,
    const uint8_t *p_src,
    uint32_t src_len,
    sgx_cmac_128bit_tag_t *p_mac
);
```

Parameters

p_key [in]

A pointer to key to be used in the CMAC hash operation. The size *must* be 128 bits.

p_src [in]

A pointer to the input data stream to be hashed. A zero length input buffer is supported, but the pointer must be non-NULL.

src_len [in]

Specifies the length on the input data stream to be hashed. A zero length input buffer is supported.

p_mac [out]

A pointer to the output 128-bit hash resulting from the CMAC calculation. This pointer must be non-NULL and the caller allocates memory for this buffer.

Return value

SGX_SUCCESS

The CMAC hash function is performed successfully.

SGX_ERROR_INVALID_PARAMETER

The key, source or MAC pointer is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An unexpected internal cryptography library.

Description

The `sgx_rijndael128_cmac_msg` function performs a standard CMAC hash over the input data buffer. Only a 128-bit version of the CMAC hash is supported.

The function should be used if the complete input data stream is available. Otherwise, the Init, Update… Update, Final procedure should be used to compute a CMAC hash over multiple input data sets.

A zero-length input data buffer is supported, but the pointer must be non-NULL.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_cmac128_init

`sgx_cmac128_init` returns an allocated and initialized CMAC algorithm context state. This should be part of the Init, Update … Update, Final process when the CMAC hash is to be performed over multiple datasets. If a complete dataset is available, the recommend call is `sgx_rijndael128_cmac_msg` to perform the hash in a single call.

Syntax

```
sgx_status_t sgx_cmac128_init(
    const sgx_cmac_128bit_key_t *p_key,
    sgx_cmac_state_handle_t* p_cmac_handle
);
```

Parameters

p_key [in]

A pointer to key to be used in the CMAC hash operation. The size *must* be 128 bits.

p_cmac_handle [out]

This is a handle to the context state used by the cryptography library to perform an iterative CMAC 128-bit hash. The algorithm stores the intermediate results of performing the hash calculation over data sets.

Return value

SGX_SUCCESS

The CMAC hash state is successfully allocated and initialized.

SGX_ERROR_INVALID_PARAMETER

The key or handle pointer is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An internal cryptography library failure occurred.

Description

Calling `sgx_cmac128_init` is the first set in performing a CMAC 128-bit hash over multiple datasets. The caller does not allocate memory for the CMAC state that this function returns. The state is specific to the implementation of the cryptography library and thus the allocation is performed by the library itself. If the hash over the desired datasets is completed or any error occurs during the hash calculation process, sgx_cmac128_close should be called to free the state allocated by this algorithm.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_cmac128_update

`sgx_cmac128_update` performs a CMAC 128-bit hash over the input dataset provided. This function supports an iterative calculation of the hash over multiple datasets where the cmac_handle contains the intermediate results of the hash calculation over previous datasets.

Syntax

```
sgx_status_t sgx_cmac128_update(
    const uint8_t *p_src,
    uint32_t src_len,
    sgx_cmac_state_handle_t cmac_handle
);
```

Parameters

p_src [in]

A pointer to the input data stream to be hashed. A zero length input buffer is supported, but the pointer must be non-NULL.

src_len [in]

Specifies the length on the input data stream to be hashed. A zero length input buffer is supported.

cmac_handle [in]

This is a handle to the context state used by the cryptography library to perform an iterative CMAC hash. The algorithm stores the intermediate results of performing the hash calculation over multiple data sets.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

The source pointer or cmac handle is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An internal cryptography library failure occurred while performing the CMAC hash calculation.

NOTE:

If an unexpected error occurs, then the CMAC state is freed (CMAC handle). In this case, call `sgx_cmac128_close` to free the CMAC state to avoid memory leak.

Description

This function should be used as part of a CMAC 128-bit hash calculation over multiple datasets. If a CMAC hash is needed over a single data set, function `sgx_rijndael128_cmac128_msg` should be used instead. Prior to calling this function on the first dataset, the `sgx_cmac128_init` function must be called first to allocate and initialize the CMAC state structure which will hold intermediate hash results over earlier datasets. The function `sgx_cmac128_final` should be used to obtain the hash after the final dataset has been processed by this function.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_cmac128_final

`sgx_cmac128_final` obtains the CMAC 128-bit hash after the final dataset has been processed (by calls to sgx_cmac128_update).

Syntax

```
sgx_status_t sgx_cmac128_final(
    sgx_cmac_state_handle_t cmac_handle,
    sgx_cmac_128bit_tag_t* p_hash
```

```
);
```

Parameters

cmac_handle [in]

This is a handle to the context state used by the cryptography library to perform an iterative CMAC hash. The algorithm stores the intermediate results of performing the hash calculation over multiple data sets.

p_hash [out]

This is a pointer to the 128-bit hash that has been calculated. The memory for the hash should be allocated by the calling code.

Return value

SGX_SUCCESS

The hash is obtained successfully.

SGX_ERROR_INVALID_PARAMETER

The hash pointer or CMAC handle is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The CMAC state passed in is likely problematic causing an internal cryptography library failure.

NOTE:

If an unexpected error occurs, then the CMAC state is freed (CMAC handle). In this case, please call sgx_cmac128_close to free the CMAC state to avoid memory leak.

Description

This function returns the hash after performing the CMAC 128-bit hash calculation over one or more datasets using the `sgx_cmac128_update` function. Memory for the hash should be allocated by the calling code. The handle to CMAC state used in the `sgx_cmac128_update` calls must be passed in as input.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_cmac128_close

`sgx_cmac128_close` cleans up and deallocates the CMAC algorithm context state that was allocated in function `sgx_cmac128_init`.

Syntax

```
sgx_status_t sgx_cmac128_close(
    sgx_cmac_state_handle_t cmac_handle
```

```
);
```

Parameters

cmac_handle [in]

This is a handle to the context state used by the cryptography library to perform an iterative CMAC hash. The algorithm stores the intermediate results of performing the hash calculation over multiple data sets.

Return value

SGX_SUCCESS

The CMAC state was deallocated successfully.

SGX_ERROR_INVALID_PARAMETER

The CMAC handle is NULL.

Description

Calling `sgx_cmac128_close` is the last step after performing a CMAC hash over multiple datasets. The caller uses this function to deallocate memory used for storing the CMAC algorithm context state.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_aes_ctr_encrypt

`sgx_aes_ctr_encrypt` performs a Rijndael AES-CTR encryption operation (counter mode). Only a 128bit key size is supported by this Intel® SGX SDK cryptography library.

Syntax

```
sgx_status_t sgx_aes_ctr_encrypt(
     const sgx_aes_ctr_128bit_key_t *p_key,
     const uint8_t *p_src,
     const uint32_t src_len,
     uint8_t *p_ctr,
     const uint32_t ctr_inc_bits,
     uint8_t *p_dst,
);
```

Parameters

p_key [in]

A pointer to key to be used in the AES-CTR encryption operation. The size *must* be 128 bits.

p_src [in]

A pointer to the input data stream to be encrypted.

src_len [in]

Specifies the length on the input data stream to be encrypted.

p_ctr [in/out]

A pointer to the initialization vector to be used in the AES-CTR calculation.

ctr_inc_bits [in]

Specifies the number of bits in the counter to be incremented.

p_dst [out]

A pointer to the output encrypted data buffer. This buffer should be allocated by the calling code.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

If key, source, destination, or counter pointer is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An internal cryptography library failure occurred.

Description

This function encrypts the input data stream of a variable length according to the CTR mode as specified in [NIST SP 800-38A]. The counter can be thought of as an IV which increments on successive encryption or decryption calls. For a given dataset or data stream, the incremented counter block should be used on successive calls of the encryption process for that given stream. However, for new or different datasets/streams, the same counter should not be reused, instead initialize the counter for the new data set.

It is recommended that the source, destination and counter data buffers are allocated within the enclave.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_aes_ctr_decrypt

`sgx_aes_ctr_decrypt` performs a Rijndael AES-CTR decryption operation (counter mode). Only a 128bit key size is supported by this Intel® SGX SDK cryptography library.

Syntax

```
sgx_status_t sgx_aes_ctr_decrypt(
    const sgx_aes_gcm_128bit_key_t *p_key,
    const uint8_t *p_src,
    const uint32_t src_len,
    uint8_t *p_ctr,
    const uint32_t ctr_inc_bits,
    uint8_t *p_dst
```

```
);
```

Parameters

p_key [in]

A pointer to key to be used in the AES-CTR decryption operation. The size *must* be 128 bits.

p_src [in]

A pointer to the input data stream to be decrypted.

src_len [in]

Specifies the length of the input data stream to be decrypted.

p_ctr [in]

A pointer to the initialization vector to be used in the AES-CTR calculation.

ctr_inc_bits [in]

Specifies the number of bits in the counter to be incremented.

p_dst [out]

A pointer to the output decrypted data buffer. This buffer should be allocated by the calling code.

Return value

SGX_SUCCESS

All the outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

If key, source, destination, or counter pointer is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An internal cryptography library failure occurred.

Description

This function decrypts the input data stream of a variable length according to the CTR mode as specified in [NIST SP 800-38A]. The counter can be thought of as an IV which increments on successive encryption or decryption calls. For a given dataset or data stream, the incremented counter block should be used on successive calls of the decryption process for that given stream. However, for new or different datasets/streams, the same counter should not be reused, instead initialize the counter for the new data set.

It is recommended that the source, destination and counter data buffers are allocated within the enclave.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_ecc256_open_context

`sgx_ecc256_open_context` returns an allocated and initialized context for the elliptic curve cryptosystem over a prime finite field, GF(p). This context must be created prior to calling `sgx_ecc256_create_key_pair` or `sgx_ecc256_compute_shared_dhkey`. When the calling code has completed its set of ECC operations, `sgx_ecc256_close_context` should be called to cleanup and deallocate the ECC context.

NOTE:

Only a field element size of 256 bits is supported.

Syntax

```
sgx_status_t sgx_ecc256_open_context(
    sgx_ecc_state_handle_t *ecc_handle
);
```

Parameters

p_ecc_handle [in]

This is a handle to the ECC GF(p) context state allocated and initialized used to perform elliptic curve cryptosystem standard functions. The algorithm stores the intermediate results of calculations performed using this context.

NOTE:

The ECC set of APIs only support a 256-bit GF(p) cryptography system.

Return value

SGX_SUCCESS

The ECC256 GF(p) state is allocated and initialized properly.

SGX_ERROR_INVALID_PARAMETER

The ECC context handle is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The ECC context state was not initialized properly due to an internal cryptography library failure.

Description

Calling `sgx_ecc256_open_context` is utilized to allocation and initialize a 256-bit GF(p) cryptographic system. The caller does not allocate memory for the ECC state that this function returns. The state is specific to the implementation of the cryptography library and thus the allocation is performed by the library itself. If the ECC cryptographic functions using this cryptographic system is completed or any error occurs, `sgx_sha256_close_context` should be called to free the state allocated by this algorithm.

Public key cryptography successfully allows to solving problems of information safety by enabling trusted communication over insecure channels. Although elliptic curves are well studied as a branch of mathematics, an interest to the cryptographic schemes based on elliptic curves is constantly rising due to the advantages that the elliptic curve algorithms provide in the wireless communications: shorter processing time and key length.

Elliptic curve cryptosystems (ECCs) implement a different way of creating public keys. As elliptic curve calculation is based on the addition of the rational points in the (x,y) plane and it is difficult to solve a discrete logarithm from these points, a higher level of safety is achieved through the cryptographic schemes that use the elliptic curves. The cryptographic systems that encrypt messages by using the properties of elliptic curves are hard to attack due to the extreme complexity of deciphering the private key.

Using of elliptic curves allows shorter public key length and encourages cryptographers to create cryptosystems with the same or higher encryption strength as the RSA or DSA cryptosystems. Because of the relatively short key length, ECCs do encryption and decryption faster on the hardware that requires less computation processing volumes.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_ecc256_close_context

`sgx_ecc256_close_context` cleans up and deallocates the ECC 256 GF(p) state that was allocated in function `sgx_ecc256_open_context`.

NOTE:

Only a field element size of 256 bits is supported.

Syntax

```
sgx_status_t sgx_ecc256_close_context(
    sgx_ecc_state_handle_t ecc_handle
);
```

Parameters

ecc_handle [in]

This is a handle to the ECC GF(p) context state allocated and initialized used to perform elliptic curve cryptosystem standard functions. The algorithm stores the intermediate results of calculations performed using this context.

NOTE:

The ECC set of APIs only support a 256-bit GF(p) cryptography system.

Return value

SGX_SUCCESS

The ECC 256 GF(p) state was deallocated successfully.

SGX_ERROR_INVALID_PARAMETER

The input handle is NULL.

Description

Calling `sgx_ecc256_close_context` is used by calling code to deallocate memory used for storing the ECC 256 GF(p) state used in ECC cryptographic calculations.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_ecc256_create_key_pair

`sgx_ecc256_create_key_pair` generates a private/public key pair on the ECC curve for the given cryptographic system. The calling code is responsible for allocating memory for the public and private keys. `sgx_ecc256_open_context` must be called to allocate and initialize the ECC context prior to making this call.

Syntax

```
sgx_status_t sgx_ecc256_create_key_pair(
    sgx_ec256_private_t *p_private,
    sgx_ec256_public_t *p_public,
    sgx_ecc_state_handle_t ecc_handle
);
```

Parameters

p_private [in]

A pointer to the private key which is a number that lies in the range of [1, n-1] where n is the order of the elliptic curve base point.

> ***NOTE:***
>
> Value is LITTLE ENDIAN.

p_public [in]

A pointer to the public key which is an elliptic curve point such that:

public key = private key * G, where G is the base point of the elliptic curve.

> ***NOTE:***
>
> Value is LITTLE ENDIAN.

ecc_handle [in]

This is a handle to the ECC GF(p) context state allocated and initialized used to perform elliptic curve cryptosystem standard functions. The algorithm stores the intermediate results of calculations performed using this context.

NOTE:

The ECC set of APIs only support a 256-bit GF(p) cryptography system.

Return value

SGX_SUCCESS

The public/private key pair was successfully generated.

SGX_ERROR_INVALID_PARAMETER

The ECC context handle, private key or public key is invalid.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The key creation process failed due to an internal cryptography library failure.

Description

This function populates private/public key pair. The calling code allocates memory for the private and public key pointers to be populated. The function generates a private key `p_private` and computes a public key `p_public` of the elliptic cryptosystem over a finite field GF(p).

The private key `p_private` is a number that lies in the range of `[1, n-1]` where n is the order of the elliptic curve base point.

The public key `p_public` is an elliptic curve point such that `p_public = p_private *G`, where `G` is the base point of the elliptic curve.

The context of the point `p_public` as an elliptic curve point must be created by using the function `sgx_ecc256_open_context`.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_ecc256_compute_shared_dhkey

`sgx_ecc256_compute_shared_dhkey` generates a secret key shared between two participants of the cryptosystem. The calling code should allocate memory for the shared key to be generated by this function.

Syntax

```
sgx_status_t sgx_ecc256_compute_shared_dhkey(
    sgx_ec256_private_t *p_private_b,
    sgx_ec256_public_t *p_public_ga,
    sgx_ec256_dh_shared_t *p_shared_key,
    sgx_ecc_state_handle_t ecc_handle
```

```
);
```

Parameters

p_private_b [in]

A pointer to the local private key.

NOTE:

Value is LITTLE ENDIAN.

---

p_public_ga [in]

A pointer to the remote public key.

NOTE:

Value is LITTLE ENDIAN.

---

p_shared_key [in]

A pointer to the secret key generated by this function which is a common point on the elliptic curve.

NOTE:

Value is LITTLE ENDIAN.

---

ecc_handle [in]

This is a handle to the ECC GF(p) context state allocated and initialized used to perform elliptic curve cryptosystem standard functions. The algorithm stores the intermediate results of calculations performed using this context.

NOTE:

The ECC set of APIs only support a 256-bit GF(p) cryptography system.

Return value

SGX_SUCCESS

The public/private key pair was successfully generated.

SGX_ERROR_INVALID_PARAMETER

The ECC context handle, private key, public key, or shared key pointer is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The key creation process failed due to an internal cryptography library failure.

Description

This function computes the Diffie-Hellman shared key based on the enclave's own (local) private key and remote enclave's public Ga Key. The calling code allocates memory for shared key to be populated by this function.

The function computes a secret number sharedKey, which is a secret key shared between two participants of the cryptosystem.

In cryptography, metasyntactic names such as Alice as Bob are normally used as examples and in discussions and stand for participant A and participant B.

Both participants (Alice and Bob) use the cryptosystem for receiving a common secret point on the elliptic curve called a secret key (sharedKey). To receive a secret key, participants apply the Diffie-Hellman key-agreement scheme involving public key exchange. The value of the secret key entirely depends on participants.

According to the scheme, Alice and Bob perform the following operations:

1. Alice calculates her own public key pubKeyA by using her private key:

privKeyA: `pubKeyA = privKeyA * G`, where `G` is the base point of the elliptic curve.

2. Alice passes the public key to Bob.

3. Bob calculates his own public key pubKeyB by using his private key

privKeyB: `pubKeyB = privKeyB * G`, where `G` is a base point of the elliptic curve.

4. Bob passes the public key to Alice.

5. Alice gets Bob's public key and calculates the secret point shareKeyA. When calculating, she uses her own private key and Bob's public key and applies the following formula:

```
shareKeyA = privKeyA * pubKeyB = privKeyA * privKeyB * G.
```

6. Bob gets Alice's public key and calculates the secret point shareKeyB. When calculating, he uses his own private key and Alice's public key and applies the following formula:

```
shareKeyB = privKeyB * pubKeyA = privKeyB * privKeyA * G.
```

As the following equation is true `privKeyA * privKeyB * G = privKeyB * privKeyA * G`, the result of both calculations is the same, that is, the equation shareKeyA = shareKeyB is true. The secret point serves as a secret key.

Shared secret shareKey is an x-coordinate of the secret point on the elliptic curve. The elliptic curve domain parameters must be hitherto defined by the function: `sgx_ecc256_open_context`.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_ecc256_check_point

`sgx_ecc256_check_point` checks whether the input point is a valid point on the ECC curve for the given cryptographic system. `sgx_ecc256_open_context` must be called to allocate and initialize the ECC context prior to making this call.

Syntax

```
sgx_status_t sgx_ecc256_check_point(
```

```
    const sgx_ec256_public_t *p_point,
    const sgx_ecc_state_handle_t ecc_handle,
    int *p_valid
);
```

Parameters

p_point [in]

A pointer to the point to perform validity check on.

NOTE:

Value is LITTLE ENDIAN.

---

ecc_handle [in]

This is a handle to the ECC GF(p) context state allocated and initialized used to perform elliptic curve cryptosystem standard functions. The algorithm stores the intermediate results of calculations performed using this context.

NOTE:

The ECC set of APIs only support a 256-bit GF(p) cryptography system.

---

p_valid [in]

A pointer to the validation result.

Return value

SGX_SUCCESS

The validation process is performed successfully. Check p_valid to get the validation result.

SGX_ERROR_INVALID_PARAMETER

If the input ecc handle, p_point or p_valid is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An internal cryptography library failure occurred.

Description

`sgx_ecc256_check_point` validates whether the input point is a valid point on the ECC curve for the given cryptographic system.

The typical validation result is one of the two values:

`1` - The input point is valid

`0` – The input point is not valid

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_ecdsa_sign

`sgx_ecdsa_sign` computes a digital signature with a given private key over an input dataset.

Syntax

```
sgx_status_t sgx_ecdsa_sign(
    const uint8_t *p_data,
    uint32_t data_size,
    sgx_ec256_private_t *p_private,
    sgx_ec256_signature_t *p_signature,
    sgx_ecc_state_handle_t ecc_handle
);
```

Parameters

p_data [in]

A pointer to the data to calculate the signature over.

data_size [in]

The size of the data to be signed.

p_private [in]

A pointer to the signature generated by this function.

NOTE:

Value is LITTLE ENDIAN.

---

p_signature [out]

A pointer to the signature generated by this function.

NOTE:

Value is LITTLE ENDIAN.

---

ecc_handle [in]

This is a handle to the ECC GF(p) context state allocated and initialized used to perform elliptic curve cryptosystem standard functions. The algorithm stores the intermediate results of calculations performed using this context.

NOTE:

The ECC set of APIs only support a 256-bit GF(p) cryptography system.

Return value

SGX_SUCCESS

The digital signature is successfully generated.

SGX_ERROR_INVALID_PARAMETER

The ECC context handle, private key, data, or signature pointer is NULL. If the data size is 0.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The signature generation process failed due to an internal cryptography library failure.

Description

This function computes a digital signature over the input dataset based on the input private key.

A message digest is a fixed size number derived from the original message with an applied hash function over the binary code of the message. (SHA256 in this case)

The signer's private key and the message digest are used to create a signature.

A digital signature over a message consists of a pair of large numbers, 256-bits each, which the given function computes.

The scheme used for computing a digital signature is of the ECDSA scheme, an elliptic curve of the DSA scheme.

The keys can be generated and set up by the function: `sgx_ecc256_create_key_pair`.

The elliptic curve domain parameters must be created by function: `sgx_ecc256_open_context`.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_ecdsa_verify

`sgx_ecdsa_verify` verifies the input digital signature with a given public key over an input data-set.

Syntax

```
sgx_status_t sgx_ecdsa_verify(
    const uint8_t *p_data,
    uint32_t data_size,
    const sgx_ec256_public_t *p_public,
    sgx_ec256_signature_t *p_signature,
    uint8_t *p_result,
    sgx_ecc_state_handle_t ecc_handle
);
```

Parameters

p_data [in]

A pointer to the signed dataset to verify.

data_size [in]

The size of the dataset to have its signature verified.

p_public [in]

A pointer to the public key to be used in the calculation of the signature.

NOTE:

Value is LITTLE ENDIAN.

---

p_signature [in]

A pointer to the signature to be verified.

NOTE:

Value is LITTLE ENDIAN.

---

p_result [out]

A pointer to the result of the verification check populated by this function.

ecc_handle [in]

This is a handle to the ECC GF(p) context state allocated and initialized used to perform elliptic curve cryptosystem standard functions. The algorithm stores the intermediate results of calculations performed using this context.

NOTE:

The ECC set of APIs only support a 256-bit GF(p) cryptography system.

Return value

SGX_SUCCESS

The digital signature verification was performed successfully. Check p_result to get the verification result.

SGX_ERROR_INVALID_PARAMETER

The ECC context handle, public key, data, result or signature pointer is NULL or the data size is 0.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The verification process failed due to an internal cryptography library failure.

Description

This function verifies the signature for the given data set based on the input public key.

A digital signature over a message consists of a pair of large numbers, 256-bits each, which could be created by function: `sgx_ecdsa_sign`. The scheme used for computing a digital signature is of the ECDSA scheme, an elliptic curve of the DSA scheme.

The typical result of the digital signature verification is one of the two values:

`SGX_ECValid` - Digital signature is valid

`SGX_ECInvalidSignature` - Digital signature is not valid

The elliptic curve domain parameters must be created by function: `sgx_ecc256_open_context`.

Requirements

Header	`sgx_tcrypto.h`
Library	`sgx_tcrypto.lib` or `sgx_tcrypto_opt.lib`

sgx_create_pse_session

`sgx_create_pse_session` creates a session with the PSE.

Syntax

```
sgx_status_t sgx_create_pse_session(
    void
);
```

Return value

SGX_SUCCESS

Session is created successfully.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond or the requested service is not supported.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_BUSY

The requested service is temporarily not available.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UPDATE_NEEDED

Intel® SGX needs to be updated.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurred.

Description

An Intel® SGX enclave first calls `sgx_create_pse_session()` in the process to request platform service.

It's suggested that the caller should wait (typically several seconds to tens of seconds) and retry this API if **SGX_ERROR_BUSY** is returned.

Requirements

Header	`sgx_tae_service.h sgx_tae_service.edl`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_close_pse_session

`sgx_close_pse_session` closes a session created by `sgx_create_pse_session`.

Syntax

```
sgx_status_t sgx_close_pse_session(
    void
);
```

Return value

SGX_SUCCESS

Session is closed successfully.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond or the requested service is not supported.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs.

Description

An Intel® SGX enclave calls `sgx_close_pse_session()` when there is no need to request platform service.

Requirements

Header	`sgx_tae_service.h sgx_tae_service.edl`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_get_ps_sec_prop

`sgx_get_ps_sec_prop` gets a data structure describing the security property of the platform service.

Syntax

```
sgx_status_t sgx_get_ps_sec_prop (
    sgx_ps_sec_prop_desc_t* security_property
);
```

Parameters

security_property [out]

A pointer to the buffer that receives the security property descriptor of the platform service. The pointer cannot be NULL.

Return value

SGX_SUCCESS

Security property is returned successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers is invalid.

SGX_ERROR_AE_SESSION_INVALID

Session is not created or has been closed by architectural enclave service.

Description

Gets a data structure that describes the security property of the platform service.

The caller should call `sgx_create_pse_session` to establish a session with the platform service enclave before calling this API.

Requirements

Header	`sgx_tae_service.h sgx_tae_service.edl`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_get_trusted_time

`sgx_get_trusted_time` gets trusted time from the AE service.

Syntax

```
sgx_status_t sgx_get_trusted_time(
    sgx_time_t* current_time,
    sgx_time_source_nonce_t* time_source_nonce
);
```

Parameters

current_time [out]

Trusted Time Stamp in seconds relative to a reference point. The reference point does not change as long as the `time_source_nonce` has not changed. The pointer cannot be NULL.

time_source_nonce [out]

A pointer to the buffer that receives the nonce which indicates time source. The pointer cannot be NULL.

Return value

SGX_SUCCESS

Trusted time is obtained successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers is invalid.

SGX_ERROR_AE_SESSION_INVALID

Session is not created or has been closed by architectural enclave service.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond or the requested service is not supported.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs.

Description

`current_time` contains time in seconds and `time_source_nonce` contains nonce associate with the time. The caller should compare `time_source_nonce` against the value returned from the previous call of this API if it needs to calculate the time passed between two readings of the Trusted Timer. If the `time_source_nonce` of the two readings do not match, the difference between the two readings does not necessarily reflect time passed.

The caller should call `sgx_create_pse_session` to establish a session with the platform service enclave before calling this API.

Requirements

Header	`sgx_tae_service.h sgx_tae_service.edl`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_create_monotonic_counter_ex

`sgx_create_monotonic_counter_ex` creates a monotonic counter.

Syntax

```
sgx_status_t sgx_create_monotonic_counter_ex(
    uint16_t owner_policy,
    const sgx_attributes_t * owner_attribute_mask,
    sgx_mc_uuid_t * counter_uuid,
    uint32_t * counter_value
);
```

Parameters

owner_policy [in]

Owner_policy of the monotonic counter.

- 0x1 means enclaves with same signing key can access the monotonic counter
- 0x2 means enclave with same measurement can access the monotonic counter
- 0x1 |0x2 means enclave with same measurement as well as signing key can access the monotonic counter.
- Owner policy values of 0x0 or any bits set beyond bits 0 and 1 will cause SGX_ERROR_INVALID_PARAMETER

owner_attribute_mask [in]

Mask of owner attribute, is the format of `sgx_attributes_t`.

counter_uuid [out]

A pointer to the buffer that receives the monotonic counter ID. The pointer cannot be NULL.

counter_value [out]

A pointer to the buffer that receives the monotonic counter value. The pointer cannot be NULL.

Return value

SGX_SUCCESS

Monotonic counter is created successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the parameters is invalid.

SGX_ERROR_MC_OVER_QUOTA

The enclave has reached the quota(256) of Monotonic Counters it can maintain.

SGX_ERROR_MC_USED_UP

Monotonic counters are used out.

SGX_ERROR_AE_SESSION_INVALID

Session is not created or has been closed by the architectural enclave service.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond or the requested service is not supported.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs.

Description

Call `sgx_create_monotonic_counter_ex()` to create a monotonic counter with the given `owner_policy` and `owner_attribute_mask`.

The caller should call `sgx_create_pse_session` to establish a session with the platform service enclave before calling this API.

NOTE

One application is not able to access the monotonic counter created by another application in simulation mode. This also affects two different applications using the same enclave.

Requirements

Header	`sgx_tae_service.h sgx_tae_service.edl`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_create_monotonic_counter

`sgx_create_monotonic_counter` creates a monotonic counter with default owner policy.

Syntax

```
sgx_status_t sgx_create_monotonic_counter(
    sgx_mc_uuid_t * counter_uuid,
    uint32_t * counter_value
);
```

Parameters

counter_uuid [out]

A pointer to the buffer that receives the monotonic counter ID. The pointer cannot be NULL.

counter_value [out]

A pointer to the buffer that receives the monotonic counter value. The pointer cannot be NULL.

Return value

SGX_SUCCESS

Monotonic counter is created successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers is invalid.

SGX_ERROR_MC_OVER_QUOTA

The enclave has reached the quota(256) of Monotonic Counters it can maintain.

SGX_ERROR_MC_USED_UP

Monotonic counters are used out.

SGX_ERROR_AE_SESSION_INVALID

Session is not created or has been closed by architectural enclave service.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond or the requested service is not supported.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs.

Description

Call `sgx_create_monotonic_counter()` to create a monotonic counter with default `owner_policy 0x1`, which means enclaves with same signing key can access the monotonic counter and default `owner_attribute_mask 0xFFFFFFFFFFFFFFCB`.

The caller should call `sgx_create_pse_session` to establish a session with the platform service enclave before calling this API.

NOTE

One application is not able to access the monotonic counter created by another application in simulation mode. This also affects two different applications using the same enclave.

Requirements

Header	`sgx_tae_service.h sgx_tae_service.edl`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_destroy_monotonic_counter

`sgx_destroy_monotonic_counter` destroys a monotonic counter created by `sgx_create_monotonic_counter`.

Syntax

```
sgx_status_t sgx_destroy_monotonic_counter(
    const sgx_mc_uuid_t * counter_uuid
);
```

Parameters

counter_uuid [in]

The monotonic counter ID will be destroyed.

Return value

SGX_SUCCESS

Monotonic counter is destroyed successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers is invalid.

SGX_ERROR_MC_NOT_FOUND

The Monotonic Counter ID is invalid.

SGX_ERROR_MC_NO_ACCESS_RIGHT

The enclave doesn't have the access right to specified Monotonic Counter.

SGX_ERROR_AE_SESSION_INVALID

Session is not created or has been closed by architectural enclave service.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond or the requested service is not supported.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs.

Description

Calling `sgx_destroy_monotonic_counter()` after a monotonic counter is not needed anymore.

The caller should call `sgx_create_pse_session` to establish a session with the platform service enclave before calling this API.

Requirements

Header	`sgx_tae_service.h` `sgx_tae_service.edl`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_increment_monotonic_counter

`sgx_increment_monotonic_counter` increments a monotonic counter value by `1`.

Syntax

```
sgx_status_t sgx_increment_monotonic_counter(
     const sgx_mc_uuid_t * counter_uuid,
     uint32_t * counter_value
);
```

Parameters

counter_uuid [in]

The Monotonic Counter ID to be incremented.

counter_value [out]

A pointer to the buffer that receives the Monotonic Counter value. The pointer cannot be NULL.

Return value

SGX_SUCCESS

Monotonic Counter is incremented successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers is invalid.

SGX_ERROR_MC_NOT_FOUND

the Monotonic Counter ID is invalid.

SGX_ERROR_MC_NO_ACCESS_RIGHT

The enclave doesn't have the access right to specified Monotonic Counter.

SGX_ERROR_AE_SESSION_INVALID

Session is not created or has been closed by architectural enclave service.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond or the requested service is not supported.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurs.

Description

Call `sgx_increment_monotonic_counter()` to increase a monotonic counter value by 1.

The caller should call `sgx_create_pse_session` to establish a session with the platform service enclave before calling this API.

Requirements

Header	`sgx_tae_service.h sgx_tae_service.edl`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_read_monotonic_counter

`sgx_read_monotonic_counter` returns the value of a monotonic counter.

Syntax

```
sgx_status_t sgx_increment_monotonic_counter(
     const sgx_mc_uuid_t * counter_uuid,
     uint32_t * counter_value
);
```

Parameters

counter_uuid [in]

The monotonic counter ID to be read.

counter_value [out]

A pointer to the buffer that receives the monotonic counter value. The pointer cannot be NULL.

Return value

SGX_SUCCESS

Monotonic counter is read successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers is invalid.

SGX_ERROR_MC_NOT_FOUND

the Monotonic Counter ID is invalid.

SGX_ERROR_AE_SESSION_INVALID

Session is not created or has been closed by the user or the Architectural Enclave service.

SGX_ERROR_SERVICE_UNAVAILABLE

The AE service did not respond or the requested service is not supported.

SGX_ERROR_SERVICE_TIMEOUT

A request to the AE service timed out.

SGX_ERROR_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_OUT_OF_EPC

There is not enough EPC memory to load one of the Architecture Enclaves needed to complete this operation.

SGX_ERROR_UNEXPECTED

Indicates an unexpected error occurred.

Description

Call `sgx_read_monotonic_counter()` to read the value of a monotonic counter.

The caller should call `sgx_create_pse_session` to establish a session with the platform service enclave before calling this API.

Requirements

Header	sgx_tae_service.h sgx_tae_service.edl
Library	sgx_tservice.lib or sgx_tservice_sim.lib (simulation)

sgx_ra_init

The sgx_ra_init function creates a context for the remote attestation and key exchange process.

Syntax

```
sgx_status_t sgx_ra_init(
    const sgx_ec256_public_t * p_pub_key,
    int b_pse,
    sgx_ra_context_t * p_context
);
```

Parameters

p_pub_key [in] (Little Endian)

The EC public key of the service provider based on the NIST P-256 elliptic curve.

b_pse [in]

If true, platform service information is needed in message 3. The caller should make sure a PSE session has been established using sgx_create_pse_session before attempting to establish a remote attestation and key exchange session involving platform service information.

p_context [out]

The output context for the subsequent remote attestation and key exchange process, to be used in sgx_ra_get_msg1 and sgx_ra_get_msg2.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates an error that the input parameters are invalid.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation, or contexts reach the limits.

SGX_ERROR_AE_SESSION_INVALID

The session is invalid or ended by the server.

SGX_ERROR_UNEXPECTED

Indicates that an unexpected error occured.

Description

This is the first API user should call for a key exchange process. The context returned from this function is used as a handle for other APIs in the key exchange library.

Requirements

Header	`sgx_tkey_exchange.h sgx_tkey_exchange.edl`
Library	`sgx_tkey_exchange.lib`

sgx_ra_get_keys

The `sgx_ra_get_keys` function is used to get the negotiated keys of a remote attestation and key exchange session. This function should only be called after the service provider accepts the remote attestation and key exchange protocol message 3 produced by `sgx_ra_get_msg2`.

Syntax

```
sgx_status_t sgx_ra_get_keys(
     sgx_ra_context_t context,
     sgx_ra_key_type_t type,
     sgx_ra_key_128_t *p_key
);
```

Parameters

context [in]

Context returned by `sgx_ra_init`.

type [in]

The type of the keys, which can be `SGX_RA_KEY_MK`, `SGX_RA_KEY_SK`, or`SGX_RA_VK`.

The returned `SGX_RA_KEY_MK`, `SGX_RA_KEY_SK` or `SGX_RA_VK` is derived from the Diffie-Hellman shared secret elliptic curve field element between the service provider and the application enclave:

```
SGX_RA_KEY_VK = AES-CMAC (0x00, gab x coordinate|| 0x03)
SGX_RA_KEY_MK = AES-CMAC (0x00, gab x coordinate|| 0x02)
SGX_RA_KEY_SK = AES-CMAC (0x00, gab x coordinate|| 0x01)
```

The AES-CMAC key used in the AES-CMAC operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format followed by one byte of 0x01, 0x02 or 0x03.

p_key [out]

The key returned.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates an error that the input parameters are invalid.

SGX_ERROR_INVALID_STATE

Indicates this API is invoked in incorrect order, it can be called only after a success session has been established. In other words, `sgx_ra_proc_msg2` should have been called and no error returned.

Description

After a successful key exchange process, this API can be used in the enclave to get specific key associated with this remote attestation and key exchange session.

Requirements

Header	`sgx_tkey_exchange.h sgx_tkey_exchange.edl`
Library	`sgx_tkey_exchange.lib`

sgx_ra_close

Call the `sgx_ra_close` function to release the remote attestation and key exchange context after the process is done and the context isn't needed anymore.

Syntax

```
sgx_status_t sgx_ra_close(
    sgx_ra_context_t context
);
```

Parameters

context [in]

Context returned by `sgx_ra_init`.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates the context is invalid.

Description

At the end of a key exchange process, the caller needs to use this API in an enclave to clear and free memory associated with this remote attestation session.

Requirements

Header	`sgx_tkey_exchange.h sgx_key_exchange.edl`
Library	`sgx_tkey_exchange.lib`

sgx_dh_init_session

Initialize DH secure session according to the caller's role in the establishment.

Syntax

```
sgx_status_t sgx_dh_init_session(
    sgx_dh_session_role_t role,
    sgx_dh_session_t * session
);
```

Parameters

role [in]

Indicates which role the caller plays in the secure session establishment.

The value of role of the initiator of the session establishment must be `SGX_DH_SESSION_INITIATOR`.

The value of role of the responder of the session establishment must be `SGX_DH_SESSION_RESPONDER`.

session [out]

A pointer to the instance of the DH session which contains entire information about session establishment.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

Return value

SGX_SUCCESS

Session is initialized successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the input parameters is incorrect.

Requirements

Header	`sgx_dh.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

.

sgx_dh_responder_gen_msg1

Generates MSG1 for the responder of DH secure session establishment and records ECC key pair in session structure.

Syntax

```
sgx_status_t sgx_dh_responder_gen_msg1(
    sgx_dh_msg1_t * msg1,
    sgx_dh_session_t * dh_session
);
```

Parameters

msg1 [out]

A pointer to an `sgx_dh_msg1_t` msg1 buffer. The buffer holding the msg1 message, which is referenced by this parameter, must be within the enclave.

The DH msg1 contains the responder's public key and report based target info.

dh_session [in/out]

A pointer that points to the instance of `sgx_dh_session_t`. The buffer holding the DH session information, which is referenced by this parameter, must be within the enclave.

NOTE

As output, the DH session structure contains the responder's public key and private key for the current session.

Return value

SGX_SUCCESS

MSG1 is generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the input parameters is incorrect.

SGX_ERROR_INVALID_STATE

The API is invoked in incorrect order or state.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

An unexpected error occurred.

Requirements

Header	`sgx_dh.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_dh_initiator_proc_msg1

The initiator of DH secure session establishment handles msg1 sent by responder and then generates msg2, and records initiator's ECC key pair in DH session structure.

Syntax

```
sgx_status_t sgx_dh_initiator_proc_msg1(
    const sgx_dh_msg1_t * msg1,
    sgx_dh_msg2_t * msg2,
    sgx_dh_session_t * dh_session
);
```

Parameters

msg1 [in]

point to dh message 1 buffer generated by session responder, and the buffer must be in enclave address space.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

msg2 [out]

point to dh message 2 buffer, and the buffer must be in enclave address space.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

dh_session [in/out]

point to dh session structure that is used during establishment, and the buffer must be in enclave address space.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

Return value

SGX_SUCCESS

msg1 is processed and msg2 is generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the input parameters is incorrect.

SGX_ERROR_INVALID_STATE

The API is invoked in incorrect order or state.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

An unexpected error occurred.

Requirements

Header	`sgx_dh.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_dh_responder_proc_msg2

The responder handles msg2 sent by initiator and then derives AEK, updates session information and generates msg3.

Syntax

```
sgx_status_t sgx_dh_responder_proc_msg2(
    const sgx_dh_msg2_t * msg2,
    sgx_dh_msg3_t * msg3,
    sgx_dh_session_t * dh_session,
    sgx_key_128bit_t * aek,
    sgx_dh_session_enclave_identity_t * initiator_identity
);
```

Parameters

msg2 [in]

point to dh message 2 buffer generated by session initiator, and the buffer must be in enclave address space.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

msg3 [out]

point to dh message 3 buffer generated by session responder in this function, and the buffer must be in enclave address space.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

dh_session [in/out]

point to dh session structure that is used during establishment, and the buffer must be in enclave address space.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

aek [out]

A pointer that points to instance of `sgx_key_128bit_t`. The aek is derived from the Diffie-Hellman shared secret elliptic curve field element between the two enclaves:

aek = AES-CMAC (0x00, gab x coordinate|| 0x01)

The AES-CMAC key used in the AES-CMAC operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format followed by one byte of 0x01.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

initiator_identity [out]

A pointer that points to instance of `sgx_dh_session_enclave_identity_t`. Identity information of initiator including isv svn, isv product id, sgx attributes, mr signer, and mr enclave. the buffer must be in enclave address space. The caller should check the identity of the peer and decide whether to trust the peer and use the aek.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

Return value

SGX_SUCCESS

msg2 is processed and msg3 is generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the input parameters is incorrect.

SGX_ERROR_INVALID_STATE

The API is invoked in incorrect order or state.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

An unexpected error occurred.

Requirements

Header	`sgx_dh.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

sgx_dh_initiator_proc_msg3

The initiator handles msg3 sent by responder and then derives AEK, updates session information and gets responder's identity information.

Syntax

```
sgx_status_t sgx_dh_initiator_proc_msg3(
    const sgx_dh_msg3_t * msg3,
    sgx_dh_session_t * dh_session,
    sgx_key_128bit_t * aek,
    sgx_dh_session_enclave_identity_t * responder_identity
);
```

Parameters

msg3 [in]

point to dh message 3 buffer generated by session responder, and the buffer must be in enclave address space.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

dh_session [in]

point to dh session structure that is used during establishment, and the buffer must be in enclave address space.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

aek [out]

A pointer that points to instance of `sgx_key_128bit_t`. The aek is derived from the Diffie-Hellman shared secret elliptic curve field element between the two enclaves:

aek = AES-CMAC (0x00, gab x coordinate|| 0x01)

The AES-CMAC key used in the AES-CMAC operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format followed by one byte of 0x01.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

---

responder_identity [out]

Identity information of responder including isv svn, isv product id, sgx attributes, mr signer, and mr enclave. the buffer must be in enclave address space. The caller should check the identity of the peer and decide whether to trust the peer and use the aek or the `msg3_body.additional_prop` field of msg3.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

Return value

SGX_SUCCESS

The function is done successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the input parameters is incorrect.

SGX_ERROR_INVALID_STATE

The API is invoked in incorrect order or state.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

An unexpected error occurred.

Requirements

Header	`sgx_dh.h`
Library	`sgx_tservice.lib` or `sgx_tservice_sim.lib` (simulation)

Types and Enumerations

This topic introduces the types and error codes in the following topics:

- Type Descriptions
- Error Codes

Type Descriptions

This topic section describes the following data types provided by the Intel® SGX:

- sgx_enclave_id_t
- sgx_launch_token_t
- sgx_exception_vector_t
- sgx_exception_type_t
- sgx_cpu_context_t
- sgx_exception_info_t
- sgx_exception_handler_t
- sgx_spinlock_t
- sgx_thread_t
- sgx_thread_mutex_t
- sgx_thread_mutexattr_t
- sgx_thread_cond_t
- sgx_thread_condattr_t
- sgx_attributes_t
- sgx_isv_svn_t
- sgx_cpu_svn_t
- sgx_key_id_t
- sgx_key_128bit_t
- sgx_key_request_t
- sgx_measurement_t
- sgx_mac_t
- sgx_report_data_t
- sgx_prod_id_t
- sgx_target_info_t
- sgx_report_body_t
- sgx_report_t
- sgx_aes_gcm_data_t
- sgx_sealed_data_t
- sgx_epid_group_id_t
- sgx_basename_t
- sgx_quote_t
- sgx_quote_sign_type_t
- sgx_spid_t
- sgx_quote_nonce_t
- sgx_time_source_nonce_t
- sgx_time_t

- sgx_ps_cap_t
- sgx_ps_sec_prop_desc_t
- sgx_mc_uuid_t
- sgx_ra_context_t
- sgx_ra_key_128_t
- sgx_ra_key_type_t
- sgx_ra_msg1_t
- sgx_ra_msg2_t
- sgx_ra_msg3_t
- sgx_ecall_get_ga_trusted_t
- sgx_ecall_get_msg3_trusted_t
- sgx_ecall_proc_msg2_trusted_t
- sgx_platform_info_t
- sgx_update_info_bit_t
- sgx_dh_msg1_t
- sgx_dh_msg2_t
- sgx_dh_msg3_t
- sgx_dh_msg3_body_t
- sgx_dh_session_enclave_identity_t
- sgx_dh_session_role_t
- sgx_dh_session_t
- sgx_device_status_t

sgx_enclave_id_t

An enclave ID, also referred to as an enclave handle. Used as a handle to an enclave by various functions.

Syntax

```
typedef uint64_t sgx_enclave_id_t;
```

Requirements

Header	sgx_eid.h

sgx_launch_token_t

An opaque type used to hold enclave license information. Used by sgx_create_enclave to initialize an enclave. The license is generated by the enclave licensing service.

See more details in Load and Unload an Enclave .

Syntax

```
typedef uint8_t sgx_launch_token_t[1024];
```

Requirements

Header	sgx_urts.h

sgx_exception_vector_t

The `sgx_exception_vector_t` enumeration contains the enclave supported exception vectors. If the exception vector is #BP, the exception type is `SGX_EXCEPTION_SOFTWARE`;

otherwise, the exception type is SGX_EXCEPTION_HARDWARE.

Syntax

```
typedef enum _sgx_exception_vector_t
{
    SGX_EXCEPTION_VECTOR_DE = 0, // DIV and DIV instructions
    SGX_EXCEPTION_VECTOR_DB = 1, // For Intel use only
    SGX_EXCEPTION_VECTOR_BP = 3, // INT 3 instruction
    SGX_EXCEPTION_VECTOR_BR = 5, // BOUND instruction
    SGX_EXCEPTION_VECTOR_UD = 6, // UD2 instruction or reserved opcode
    SGX_EXCEPTION_VECTOR_MF = 16, // x87 FPU floating-point or WAIT/FWAI
    instruction.
    SGX_EXCEPTION_VECTOR_AC = 17, // Any data reference in memory
    SGX_EXCEPTION_VECTOR_XM = 19, // SSE/SSE2/SSE3 floating-point instruc-
    tion
} sgx_exception_vector_t;
```

Requirements

Header	sgx_trts_exception.h

sgx_exception_type_t

The sgx_exception_type_t enumeration contains values that specify the exception type. If the exception vector is #BP (BreakPoint), the exception type is SGX_EXCEPTION_SOFTWARE; otherwise, the exception type is SGX_EXCEPTION_HARDWARE.

Syntax

```
typedef enum _sgx_exception_type_t
{
    SGX_EXCEPTION_HARDWARE = 3,
    SGX_EXCEPTION_SOFTWARE = 6,
} sgx_exception_type_t;
```

Requirements

Header	sgx_trts_exception.h

sgx_cpu_context_t

The sgx_cpu_content_t structure contains processor-specific register data. Custom exception handling uses sgx_cpu_context_t structure to record the CPU context at exception time.

Syntax

```
#if defined (_M_X64) || defined (__x86_64__)
    typedef struct _cpu_context_t
    {
        uint64_t rax;
        uint64_t rcx;
        uint64_t rdx;
        uint64_t rbx;
        uint64_t rsp;
        uint64_t rbp;
        uint64_t rsi;
```

```
        uint64_t rdi;
        uint64_t r8;
        uint64_t r9;
        uint64_t r10;
        uint64_t r11;
        uint64_t r12;
        uint64_t r13;
        uint64_t r14;
        uint64_t r15;
        uint64_t rflags;
        uint64_t rip;
    } sgx_cpu_context_t;
#else
    typedef struct _cpu_context_t
    {
        uint32_t eax;
        uint32_t ecx;
        uint32_t edx;
        uint32_t ebx;
        uint32_t esp;
        uint32_t ebp;
        uint32_t esi;
        uint32_t edi;
        uint32_t eflags;
        uint32_t eip;
    } sgx_cpu_context_t;
#endif
```

Members

rax, rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8 – r15

64-bit general purpose registers

rflags

64-bit program status and control register

rip

64-bit instruction pointer

eax, ecx, edx, ebx, esp, ebp, esi, edi

32-bit general purpose registers

eflags

32-bit program status and control register

eip

32-bit instruction pointer

Requirements

Header	sgx_trts_exception.h

sgx_exception_info_t

A structure of this type contains an exception record with a description of the exception and processor context record at the time of exception.

Syntax

```
typedef struct _exception_info_t
{
    sgx_exception_vector_t exception_vector;
    sgx_exception_type_t exception_type;
    sgx_cpu_context_t cpu_context;
} sgx_exception_info_t;
```

Members

exception_vector

The reason the exception occurs. This is the code generated by a hardware exception.

exception_type

The exception type. 3 indicating a HW exception, 6 indicating a SW exception.

cpu_context

The context record that contains the processor context at the exception time.

Requirements

Header	`sgx_trts_exception.h`

sgx_exception_handler_t

Callback function that serves as a custom exception handler.

Syntax

```
typedef int (* sgx_exception_handler_t) (sgx_exception_info_t *info);
```

Members

info

A pointer to `sgx_exception_info_t` structure that receives the exception information.

Requirements

Header	`sgx_trts_exception.h`

sgx_spinlock_t

Data type for a trusted spin lock.

Syntax

```
typedef volatile uint32_t sgx_spinlock_t;
```

Members

`sgx_spinlock_t` defines a spin lock object inside the enclave.

Requirements

Header	`sgx_spinlock.h`

sgx_thread_t

Data type to uniquely identify a trusted thread.

Syntax

```
typedef uintptr * sgx_thread_t;
```

Members

`sgx_thread_t` is an opaque data type with no member fields visible to users. This data type is subject to change. Thus, enclave code should not rely on the contents of this data object.

Requirements

Header	`sgx_thread.h`

sgx_thread_mutex_t

Data type for a trusted mutex object.

Syntax

```
typedef struct sgx_thread_mutex
{
     size_t m_refcount;
     uint32_t m_control;
     volatile uint32_t m_lock;
     sgx_thread_t m_owner;
     sgx_thread_queue_t m_queue;
} sgx_thread_mutex_t;
```

Members

m_control

Flags to define whether a mutex is recursive or not.

m_refcount

Reference counter of the mutex object. It will be increased by 1 if the mutex is successfully acquired, and be decreased by 1 if the mutex is released.

NOTE

The counter will be greater than one only if the mutex is recursive.

---

m_lock

The spin lock used to guarantee atomic updates to the mutex object.

m_owner

The thread that currently owns the mutex writes its unique thread identifier in this field, which otherwise is NULL. This field is used for error checking, for instance to ensure that only the owner of a mutex releases it.

m_queue

Ordered list of threads waiting to acquire the ownership of the mutex. The queue itself is a structure containing a head and a tail for quick insertion and removal under FIFO semantics.

Requirements

Header	sgx_thread.h

sgx_thread_mutexattr_t

Attribute for the trusted mutex object.

Syntax

```
typedef struct sgx_thread_mutex_attr
{
    unsigned char m_dummy;
} sgx_thread_mutexattr_t;
```

Members

m_dummy

Dummy member not supposed to be used.

Requirements

Header	sgx_thread.h

sgx_thread_cond_t

Data type for a trusted condition variable.

Syntax

```
typedef struct sgx_thread_cond
{
    sgx_spinlock_t m_lock;
    sgx_thread_queue_t m_queue;
} sgx_thread_cond_t;
```

Members

m_lock

The spin lock used to guarantee atomic updates to the condition variable.

m_queue

Ordered list of threads waiting on the condition variable. The queue itself is a structure containing a head and a tail for quick insertion and removal under FIFO semantics.

Requirements

Header	sgx_thread.h

sgx_thread_condattr_t

Attribute for the trusted condition variable.

Syntax

```
typedef struct sgx_thread_cond_attr
{
    unsigned char m_dummy;
} sgx_thread_condattr_t;
```

Members

m_dummy

Dummy member not supposed to be used.

Requirements

Header	sgx_thread.h

sgx_misc_select_t

Enclave misc select bits. The value is 4 byte in length. Currently all the bits are reserved for future extension.

Requirements

Header	sgx_attributes.h

sgx_attributes_t

Enclave attributes definition structure.

NOTE

When specifying an attributes mask used in key derivation, at a minimum the flags that should be set are INITED, DEBUG and RESERVED bits.

NOTE

The XGETBV instruction can be executed to determine the register sets which are part of the XSAVE state which corresponds to the xfrm value of attributes. Since the save state is dependent on the host system and operating system, an attributes mask generally does not include these bits (XFRM is set to 0).

Syntax

```
typedef struct _sgx_attributes_t
{
    uint64_t flags;
    uint64_t xfrm;
} sgx_attributes_t;
```

Members

flags

Flags is a combination of the following values.

Value	Description
SGX_FLAGS_INITTED 0x0000000000000001ULL	The enclave is initialized
SGX_FLAGS_DEBUG 0x0000000000000002ULL	The enclave is a debug enclave
SGX_FLAGS_MODE64BIT 0x0000000000000004ULL	The enclave runs in 64 bit mode
SGX_FLAGS_PROVISION_KEY 0x0000000000000010ULL	The enclave has access to a provision key
SGX_FLAGS_LICENSE_KEY 0x0000000000000020ULL	The enclave has access to a license key
SGX_FLAGS_RESERVED 0xFFFFFFFFFFFFFFC8ULL	A mask used to ensure that reserved bits are zero. Reserved bits are bit 3 and bits 6-63.

xfrm

Similar to XCR0, xfrm is a combination of the following values.

Value	Description
SGX_XFRM_LEGACY 0x0000000000000003ULL	FPU and SSE states are saved
SGX_XFRM_AVX 0x0000000000000006ULL	AVX state is saved

Requirements

Header sgx_attributes.h

sgx_misc_attribute_t

Enclave `misc_select` and attributes definition structure.

Syntax

```
typedef struct _sgx_misc_attributes_t
{
     sgx_attributes_t secs_attr;
     sgx_misc_select_t misc_select;
} sgx_misc_attribute_t;
```

Members

secs_attr

The Enclave attributes.

misc_select

The Enclave misc select configuration.

Requirements

Header	sgx_attributes.h

sgx_isv_svn_t

ISV security version. The value is 2 bytes in length. Use this value in key derivation and obtain it by getting an enclave report (`sgx_create_report`).

Requirements

Header	`sgx_key.h`

sgx_cpu_svn_t

`sgx_cpu_svn_t` is a 128-bit value representing the CPU security version. Use this value in key derivation and obtain it by getting an enclave report (`sgx_create_report`).

Syntax

```
#define SGX_CPUSVN_SIZE 16
typedef struct _sgx_cpu_svn_t {
    uint8_t svn[SGX_CPUSVN_SIZE];
} sgx_cpu_svn_t;
```

Requirements

Header	`sgx_key.h`

sgx_key_id_t

`sgx_key_id_t` is a 256 bit value used in the key request structure. The value is generally populated with a random value to provide key wear-out protection.

Syntax

```
#define SGX_KEYID_SIZE 32
typedef struct _sgx_key_id_t {
    uint8_t id[SGX_KEYID_SIZE];
} sgx_key_id_t;
```

Requirements

Header	`sgx_key.h`

sgx_key_128bit_t

A 128 bit value that is the used to store a derived key from for example the `sgx_get_key` function.

Requirements

Header	`sgx_key.h`

sgx_key_request_t

Data structure of key request which is used for selecting the appropriate key and any additional parameters required in the derivation of that key. This is the input parameter for the `sgx_get_key` function.

Syntax

```
typedef struct _key_request_t {
     uint16_t key_name;
     uint16_t key_policy;
     sgx_isv_svn_t isv_svn;
     uint16_t reserved1;
     sgx_cpu_svn_t cpu_svn;
     sgx_attributes_t attribute_mask;
     sgx_key_id_t key_id;
     sgx_misc_select_t misc_mask;
     uint8_t reserved2[436];
} sgx_key_request_t;
```

Members

key_name

The name of the key requested. Possible values are below:

Key Name	Value	Description
KEYSELECT_LICENSE	0x0000	License key
KEYSELECT_PROVISION	0x0001	Provisioning key
KEYSELECT_ PROVISION_SEAL	0x0002	Provisioning seal key
KEYSELECT_REPORT	0x0003	Report key
KEYSELECT_SEAL	0x0004	Seal key

key_policy

Identify which inputs are required to be used in the key derivation. Possible values are below:

Key policy name	Value	Description
KEYPOLICY_MRENCLAVE	0x0001	Derive key using the enclave's ENCLAVE measurement register
KEYPOLICY_MRSIGNER	0x0002	Derive key using the enclave's SIGNER measurement register

NOTE

If MRENCLAVE is used, then that key can only be rederived by that particular enclave.

NOTE

If MRSIGNER is used, then another enclave with the same ISV_SVN could derive the key as well which is useful for applications that instantiate more than one enclave and would like to pass data. The key derived could be used in the encryption process for the data passed between the enclaves.

isv_svn

The ISV security version number that should be used in the key derivation.

reserved1

Reserved for future use. Must be zero.

cpu_svn

The TCB security version number that should be used in the key derivation.

attribute_mask

The attributes mask used to determine which enclave attributes must be included in the key. It only impacts the derivation of seal key, provisioning key and provisioning seal key. See the definition of sgx_attributes_t.

key_id

Value for key wear-out protection. Generally initialized with a random number.

misc_maks

The misc mask used to determine which enclave misc select must be included in the key. Reserved for future function extension.

reserved2

Reserved for future use. Must be set to zero.

Requirements

Header	`sgx_key.h`

sgx_measurement_t

`sgx_measurement_t` is a 256-bit value representing the enclave measurement.

Syntax

```
#define SGX_HASH_SIZE 32
typedef struct _sgx_measurement_t {
    uint8_t m[SGX_HASH_SIZE];
} sgx_measurement_t;
```

Requirements

Header	`sgx_report.h`

sgx_mac_t

This type is utilized as storage for the 128-bit CMAC value of the report data.

Requirements

Header	`sgx_report.h`

sgx_report_data_t

`sgx_report_data_t` is a 512-bit value used for communication between the enclave and the target enclave. This is one of the inputs to the `sgx_create_report` function.

Syntax

```
#define SGX_REPORT_DATA_SIZE 64
typedef struct _sgx_report_data_t {
    uint8_t d[SGX_REPORT_DATA_SIZE];
} sgx_report_data_t;
```

Requirements

Header	sgx_report.h

sgx_prod_id_t

A 16-bit value representing the ISV enclave product ID. This value is used in the derivation of some keys.

Requirements

Header	sgx_report.h

sgx_target_info_t

Data structure of report target information. This is an input to function `sgx_create_report` and `sgx_init_quote` which is used to identify the enclave (its measurement and attributes) which will be able to verify the REPORT that is generated.

Syntax

```
typedef struct _targe_info_t
{
    sgx_measurement_t mr_enclave;
    sgx_attributes_t attributes;
    uint8_t reserved1[4];
    sgx_misc_select_t misc_select;
    uint8_t reserved2[456];
} sgx_target_info_t;
```

Members

mr_enclave

The enclave hash of the target enclave

attributes

The attributes of the target enclave

reserved1

Reserved for future use. Must be set to zero.

misc_select

The misc select bits for the target enclave. Reserved for future function extension.

reserved2

Reserved for future use. Must be set to zero.

Requirements

Header	sgx_report.h

sgx_report_body_t

This data structure, which is part of the `sgx_report_t` structure, contains information about the enclave.

Syntax

```
typedef struct _report_body_t
{
    sgx_cpu_svn_t cpu_svn;
    sgx_misc_select_t misc_select;
    uint8_t reserved1[28];
    sgx_attributes_t attributes;
    sgx_measurement_t mr_enclave;
    uint8_t reserved2[32];
    sgx_measurement_t mr_signer;
    uint8_t reserved3[96];
    sgx_prod_id_t isv_prod_id;
    sgx_isv_svn_t isv_svn;
    uint8_t reserved4[60];
    sgx_report_data_t report_data;
} sgx_report_body_t;
```

Members

cpu_svn

The security version number of the host system TCB (CPU).

misc_select

The misc select bits for the target enclave. Reserved for future function extension.

reserved1

Reserved for future use. Must be set to zero.

attributes

The attributes for the enclave. See sgx_attributes_t for the definitions of these flags.

mr_enclave

The measurement value of the enclave.

reserved2

Reserved for future use. Must be set to zero.

mr_signer

The measurement value of the public key that verified the enclave.

reserved3

Reserved for future use. Must be set to zero.

isv_prod_id

The ISV Product ID of the enclave.

isv_svn

The ISV security version number of the enclave.

reserved4

Reserved for future use. Must be set to zero.

report_data

A set of data used for communication between the enclave and the target enclave.

Requirements

Header	`sgx_report.h`

sgx_report_t

Data structure that contains the report information for the enclave. This is the output parameter from the `sgx_create_report` function. This is the input parameter for the `sgx_init_quote` function.

Syntax

```
typedef struct _report_t
{
    sgx_report_body_t body;
    sgx_key_id_t key_id;
    sgx_mac_t mac;
} sgx_report_t;
```

Members

body

The data structure containing information about the enclave.

key_id

Value for key wear-out protection.

mac

The CMAC value of the report data using report key.

Requirements

Header	`sgx_report.h`

sgx_aes_gcm_data_t

The structure contains the AES GCM* data, payload size, MAC* and payload.

Syntax

```
typedef struct _aes_gcm_data_t
{
    uint32_t payload_size;
    uint8_t reserved[12];
    uint8_t payload_tag[SGX_SEAL_TAG_SIZE];
    uint8_t payload[];
} sgx_aes_gcm_data_t;
```

Members

payload_size

Size of the payload data which includes both the encrypted data followed by the additional authenticated data (plain text). The full payload array is part of the AES GCM MAC calculation.

reserved

Padding to allow the data to be 16 byte aligned.

payload_tag

AES-GMAC of the plain text, payload, and the sizes

payload

The payload data buffer includes the encrypted data followed by the optional additional authenticated data (plain text),which is not encrypted.

NOTE

The optional additional authenticated data (MAC or plain text) could be data which identifies the seal data blob and when it was created.

Requirements

Header	`sgx_tseal.h`

sgx_sealed_data_t

Sealed data blob structure containing the key request structure used in the key derivation. The data structure has been laid out to achieve 16 byte alignment. This structure should be allocated within the enclave when the seal operation is performed. After the seal operation, the structure can be copied outside the enclave for preservation before the enclave is destroyed. The `sealed_data` structure needs to be copied back within the enclave before unsealing.

Syntax

```
typedef struct _sealed_data_t
{
    sgx_key_request_t key_request;
    uint32_t plain_text_offset;
    uint8_t reserved[12];
    sgx_aes_gcm_data_t aes_data;
} sgx_sealed_data_t;
```

Members

key_request

The key request used to derive the seal key.

plain_text_offset

The offset within the `aes_data` structure payload to the start of the optional additional MAC text.

reserved

Padding to allow the data to be 16 byte aligned.

aes_data

Structure contains the AES GCM data (payload size, MAC, and payload).

Requirements

Header	`sgx_tseal.h`

sgx_epid_group_id_t

Type for EPID group id

Syntax

```
typedef uint8_t sgx_epid_group_id_t[4];
```

Requirements

Header	sgx_quote.h

sgx_basename_t

Type for base name used in `sgx_quote`.

Syntax

```
typedef struct _basename_t
{
    uint8_t name[32];
} sgx_basename_t;
```

Members

name

The base name used in `sgx_quote`.

Requirements

Header	sgx_quote.h

sgx_quote_t

Type for quote used in remote attestation.

Syntax

```
typedef struct _quote_t
{
    uint16_t version;
    uint16_t sign_type;
    sgx_epid_group_id_t epid_group_id;
    sgx_isv_svn_t qe_svn;
    uint8_t reserved[6];
    sgx_basename_t basename;
    sgx_report_body_t report_body;
    uint32_t signature_len;
    uint8_t signature[];
} sgx_quote_t;
```

Members

version

The version of the quote structure.

sign_type

The indicator of the EPID signature type.

epid_group_id

The EPID group id of the platform belongs to.

qe_svn

The svn of the QE.

reserved

The reserved field of `sgx_quote_t`, used to keep structure aligned.

basename

The base name used in sgx_quote.

report_body

The report body of the application enclave.

signature_len

The size in byte of the following signature.

signature

The place holder of the variable length signature.

Requirements

Header	`sgx_quote.h`

sgx_quote_sign_type_t

Enum indicates the quote type, linkable or un-linkable

Syntax

```
typedef enum {
    SGX_UNLINKABLE_SIGNATURE,
    SGX_LINKABLE_SIGNATURE
} sgx_quote_sign_type_t;
```

Requirements

Header	`sgx_quote.h`

sgx_spid_t

Type for a service provider ID.

Syntax

```
typedef struct _spid_t
{
    uint8_t id[16];
} sgx_spid_t;
```

Members

id

The ID of the service provider.

Requirements

Header	sgx_quote.h

sgx_quote_nonce_t

This data structure indicates the quote nonce.

Syntax

```
typedef struct _sgx_quote_nonce
{
    uint8_t rand[16];
} sgx_quote_nonce_t;
```

Members

rand

The 16 bytes random number used as nonce.

Requirements

Header	sgx_quote.h

sgx_time_source_nonce_t

Nonce of time source. It’s opaque to users.

Syntax

```
typedef uint8_t sgx_time_source_nonce_t[32];
```

Requirements

Header	sgx_tae_service.h

sgx_time_t

Type for trusted time.

Syntax

```
typedef uint64_t sgx_time_t;
```

Requirements

Header	sgx_tae_service.h

sgx_ps_cap_t

Type indicating the platform service capability.

Syntax

```
typedef struct _sgx_ps_cap_t
{
    uint32_t ps_cap0;
    uint32_t ps_cap1;
```

```
} sgx_ps_cap_t;
```

Members

ps_cap0

Bit 0 : Trusted Time service

Bit 1 : Monotonic Counter service

Bit 2-31 : Reserved

ps_cap1

Bit 0-31 : Reserved

Requirements

Header	`sgx_uae_service.h`

sgx_ps_sec_prop_desc_t

Security property descriptor of platform service. It's opaque to users.

Syntax

```
typedef struct _ps_sec_prop_desc
{
    uint8_t sgx_ps_sec_prop_desc[256];
} sgx_ps_sec_prop_desc_t;
```

Requirements

Header	`sgx_tae_service.h`

sgx_mc_uuid_t

The data structure of a monotonic counter.

Syntax

```
#define SGX_MC_UUID_COUNTER_ID_SIZE 3
#define SGX_MC_UUID_NONCE_SIZE 13
typedef struct _mc_uuid
{
    uint8_t counter_id[SGX_MC_UUID_COUNTER_ID_SIZE];
    uint8_t nonce[SGX_MC_UUID_NONCE_SIZE];
} sgx_mc_uuid_t;
```

Members

counter_id

ID number of the monotonic counter.

nonce

Nonce associated with the monotonic counter.

Requirements

Header	sgx_tae_service.h

sgx_ra_context_t

Type for a context returned by the key exchange library.

Syntax

```
typedef uint32_t sgx_ra_context_t;
```

Requirements

Header	sgx_key_exchange.h

sgx_ra_key_128_t

Type for 128 bit key used in remote attestation.

Syntax

```
typedef uint8_t sgx_ra_key_128_t[16];
```

Requirements

Header	sgx_key_exchange.h

sgx_ra_key_type_t

Enum of the key types used in remote attestation.

Syntax

```
typedef enum _sgx_ra_key_type_t
{
    SGX_RA_KEY_SK = 1,
    SGX_RA_KEY_MK,
    SGX_RA_KEY_VK,
} sgx_ra_key_type_t;
```

Requirements

Header	sgx_key_exchange.h

sgx_ra_msg1_t

This data structure describes the message 1 that is used in remote attestation and key exchange protocol.

Syntax

```
typedef struct _sgx_ra_msg1_t
{
    sgx_ec256_public_t g_a;
    sgx_epid_group_id_t gid;
} sgx_ra_msg1_t;
```

Members

g_a (Little Endian)

The public EC key of an application enclave, based on NIST P-256 elliptic curve.

gid (Little Endian)

ID of the EPID group of the platform belongs to.

Requirements

Header	`sgx_key_exchange.h`

sgx_ra_msg2_t

This data structure describes the message 2 that is used in the remote attestation and key exchange protocol.

Syntax

```
typedef struct _sgx_ra_msg2_t
{
    sgx_ec256_public_t g_b;
    sgx_spid_t spid;
    sgx_quote_sign_type_t quote_type;
    sgx_ec256_signature_t sign_gb_ga;
    uint8_t mac[16];
    uint32_t sig_rl_size;
    uint8_t sig_rl[];
} sgx_ra_msg2_t;
```

Members

g_b (Little Endian)

Public EC key of service provider, based on the NIST P-256 elliptic curve.

spid

ID of the service provider

quote_type

Enum indicates the quote type, linkable or un-linkable

sign_gb_ga (Litte Endian)

ECDSA Signature of (g_b||g_a), using the service provider's ECDSA private key corresponding to the public key specified in `sgx_ra_init` function, where g_b is the public EC key of the service provider and g_a is the public key of application enclave, provided by the application enclave, in the remote attestation and key exchange message 1.

mac

AES-CMAC of gb, spid and `sign_gb_ga`, using SMK as the AES-CMAC key. SMK is derived from the Diffie-Hellman shared secret elliptic curve field element between the service provider and the application enclave:

SMK = AES-CMAC (0x00, gab x coordinate|| 0x00)

The AES-CMAC key used in the AES-CMAC operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format followed by one byte of 0x00.

sig_rl_size

Size of the `sig_rl`, in bytes.

sig_rl

Pointer to the EPID Signature Revocation List Certificate of the EPID group identified by the gid in the remote attestation and key exchange message 1.

Requirements

Header	`sgx_key_exchange.h`

sgx_ra_msg3_t

This data structure describes message 3 that is used in the remote attestation and key exchange protocol.

Syntax

```
typedef struct _sgx_ra_msg3_t
{
    uint8_t mac[16];
    sgx_ec256_public_t g_a;
    sgx_ps_sec_prop_desc_t ps_sec_prop;
    uint8_t quote[];
} sgx_ra_msg3_t;
```

Members

mac

AES-CMAC of g_a, ps_sec_prop and quote[], using SMK. SMK is derived from the Diffie-Hellman shared secret elliptic curve field element between the service provider and the application enclave:

SMK = AES-CMAC (0x00, gab x coordinate|| 0x00)

The AES-CMAC key used in the AES-CMAC operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format followed by one byte of 0x00.

g_a (Little Endian)

Public EC key of application enclave

ps_sec_prop

Security property of the Intel® SGX Platform Service. If the Intel® SGX Platform Service security property information is not required in the remote attestation and key exchange process, this field will be all 0s.

quote

Quote returned from `sgx_get_quote`. The first 32-byte `report_body.report_data` field in Quote is set to SHA256 hash of ga, gb and VK, and the second 32-byte is set to all 0s. VK is derived from the Diffie-Hellman shared secret elliptic curve field element between the service provider and the application enclave:

VK = AES-CMAC (0x00, gab x coordinate|| 0x03)

The AES-CMAC key used in the AES-CMAC operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format followed by one byte of 0x03.

Requirements

Header	sgx_key_exchange.h

sgx_ecall_get_ga_trusted_t

Function pointer of proxy function generated from sgx_tkey_exchange.edl.

Syntax

```
typedef sgx_status_t (* sgx_ecall_get_ga_trusted_t)(
     sgx_enclave_id_t eid,
     int* retval,
     sgx_ra_context_t context,
     sgx_ec256_public_t *g_a // Little Endian
);
```

Note that the 4th parameter this function takes should be in little endian format.

Requirements

Header	sgx_ukey_exchange.h

sgx_ecall_proc_msg2_trusted_t

Function pointer of proxy function generated from sgx_tkey_exchange.edl.

Syntax

```
typedef sgx_status_t (* sgx_ecall_proc_msg2_trusted_t)(
     sgx_enclave_id_t eid,
     int* retval,
     sgx_ra_context_t context,
     const sgx_ra_msg2_t *p_msg2,
     const sgx_target_info_t *p_qe_target,
     sgx_report_t *p_report,
     sgx_quote_nonce_t *p_nonce
);
```

Requirements

Header	sgx_ukey_exchange.h

sgx_ecall_get_msg3_trusted_t

Function pointer of proxy function generated from sgx_tkey_exchange.edl.

Syntax

```
typedef sgx_status_t (* sgx_ecall_get_msg3_trusted_t)(
     sgx_enclave_id_t eid,
     int* retval,
     sgx_ra_context_t context,
     uint32_t quote_size,
     sgx_report_t* qe_report,
     sgx_ra_msg3_t *p_msg3,
     uint32_t msg3_size
);
```

Requirements

Header	sgx_ukey_exchange.h

sgx_platform_info_t

This opaque data structure indicates the platform information received from Intel Attestation Server.

Syntax

```
#define SGX_PLATFORM_INFO_SIZE 290
typedef struct _platform_info
{
    uint8_t platform_info[SGX_PLATFORM_INFO_SIZE];
} sgx_platform_info_t;
```

Members

platform_info

The platform information.

Requirements

Header	sgx_quote.h

sgx_update_info_bit_t

Type for information of what components of SGX need to be updated and how to update them.

Syntax

```
typedef struct _update_info_bit
{
    int ucodeUpdate;
    int csmeFwUpdate;
    int pswUpdate;
} sgx_update_info_bit_t;
```

Members

ucodeUpdate

Whether the ucode needs to be updated.

csmeFwUpdate

Whether the csme firmware needs to be updated.

pswUpdate

Whether the platform software needs to be updated.

Requirements

Header	sgx_quote.h

sgx_dh_msg1_t

Type for MSG1 used in DH secure session establishment.

Syntax

```
typedef struct _sgx_dh_msg1_t
{
     sgx_ec256_public_t g_a;
     sgx_target_info_t target;
} sgx_dh_msg1_t;
```

Members

g_a (Little Endian)

Public EC key of responder enclave of DH session establishment, based on the NIST P-256 elliptic curve.

target

Report target info to be used by the peer enclave to generate the Intel® SGX report in the message 2 of the DH secure session protocol.

Requirements

Header	`sgx_dh.h`

sgx_dh_msg2_t

Type for MSG2 used in DH secure session establishment.

Syntax

```
typedef struct _sgx_dh_msg2_t
{
     sgx_ec256_public_t g_b;
     sgx_report_t report;
     uint8_t cmac[SGX_DH_MAC_SIZE];
} sgx_dh_msg2_t;
```

Members

g_b (Little Endian)

Public EC key of initiator enclave of DH session establishment, based on the NIST P-256 elliptic curve.

report

Intel® SGX report of initiator enclave of DH session establishment. The first 32-byte of the report_data field of the report is set to SHA256 hash of g_a and g_b, where g_a is the EC Public key of the responder enclave and g_b is the EC public key of the initiator enclave. The second 32-byte of the report_data field is set to all 0s.

cmac[SGX_DH_MAC_SIZE]

AES-CMAC value of `g_b` and report, using SMK as the AES-CMAC key. SMK is derived from the Diffie-Hellman shared secret elliptic curve field element between the two enclaves:

SMK = AES-CMAC (0x00, g^{ab} x coordinate|| 0x00)

The AES-CMAC key used in the AES-CMAC operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format followed by one byte of 0x00.

Requirements

Header	sgx_dh.h

sgx_dh_msg3_t

Type for MSG3 used in DH secure session establishment.

Syntax

```
typedef struct _sgx_dh_msg3_t
{
    uint8_t cmac[SGX_DH_MAC_SIZE];
    sgx_dh_msg3_body_t msg3_body;
} sgx_dh_msg3_t;
```

Members

cmac[SGX_DH_MAC_SIZE]

CMAC value of message body of MSG3, using SMK as the AES-CMAC key. SMK is derived from the Diffie-Hellman shared secret elliptic curve field element between the two enclaves:

SMK = AES-CMAC (0x00, g^{ab} x coordinate|| 0x00)

The AES-CMAC key used in the AES-CMAC operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format followed by one byte of 0x00.

msg3_body

Variable length message body of MSG3.

Requirements

Header	sgx_dh.h

sgx_dh_msg3_body_t

Type for message body of the MSG3 structure used in DH secure session establishment.

Syntax

```
typedef struct _sgx_dh_msg3_body_t
{
    sgx_report_t report;
    uint32_t additional_prop_length;
    uint8_t additional_prop[0];
} sgx_dh_msg3_body_t;
```

Members

report

Intel® SGX report of responder enclave. The first 32-byte of the report_data field of the report is set to SHA256 hash of g_b and g_a, where g_a is the EC Public key of the responder enclave and g_b is the EC public key of the initiator enclave. The second 32-byte of the report_data field is set to all 0s.

additional_prop_length

Length of additional property field in bytes.

additional_prop[0]

Variable length buffer holding additional data that the responder enclave may provide.

Requirements

Header	sgx_dh.h

sgx_dh_session_enclave_identity_t

Type for enclave identity of initiator or responder used in DH secure session establishment.

Syntax

```
typedef struct _sgx_dh_session_enclave_identity_t
{
     sgx_cpu_svn_t cpu_svn;
     uint8_t reserved_1[32];
     sgx_attributes_t attributes;
     sgx_measurement_t mr_enclave;
     uint8_t reserved_2[32];
     sgx_measurement_t mr_signer;
     uint8_t reserved_3[96];
     sgx_prod_id_t isv_prod_id;
     sgx_isv_svn_t isv_svn;
} sgx_dh_session_enclave_identity_t;
```

Members

cpu_svn

Security version number of CPU.

reserved_1[32]

Reserved 32 bytes.

attributes

SGX attributes of enclave.

mr_enclave

Measurement of enclave.

reserved_2[32]

Reserved 32 bytes.

mr_signer

Measurement of enclave signer.

reserved_3[96]

Reserved 96 bytes.

isv_prod_id (Little Endian)

Product ID of ISV enclave.

isv_svn (Little Endian)

Security version number of ISV enclave.

Requirements

Header	sgx_dh.h

sgx_dh_session_role_t

Type for role of establishing a DH secure session used in DH secure session establishment.

Syntax

```
typedef enum _sgx_dh_session_role_t
{
    SGX_DH_SESSION_INITIATOR,
    SGX_DH_SESSION_RESPONDER
} sgx_dh_session_role_t;
```

Members

SGX_DH_SESSION_INITIATOR

Initiator of a DH session establishment.

SGX_DH_SESSION_RESPONDER

Responder of a DH session establishment.

Requirements

Header	sgx_dh.h

sgx_dh_session_t

Type for session used in DH secure session establishment.

Syntax

```
typedef struct _sgx_dh_session_t
{
    uint8_t sgx_dh_session[SGX_DH_SESSION_DATA_SIZE];
} sgx_dh_session_t;
```

Members

sgx_dh_session

Data of DH session.

The array size of sgx_dh_session SGX_DH_SESSION_DATA_SIZE is defined as 200 bytes.

Requirements

Header	sgx_dh.h

sgx_device_status_t

Type for the status of Intel® SGX device after the dynamic enabling.

Syntax

```
typedef enum _sgx_device_status_t
{
    SGX_ENABLED,
    SGX_DISABLED_REBOOT_REQUIRED,
```

```
    SGX_DISABLED_LEGACY_OS,
    SGX_DISABLED,
    SGX_DISABLED_SCI_AVAILABLE
} sgx_device_status_t;
```

Members

SGX_ENABLED

Intel SGX device is enabled.

SGX_DISABLED_REBOOT_REQUIRED

Intel SGX device is disabled and a reboot is required to enable it.

SGX_DISABLED_LEGACY_OS

The operating system is a legacy system and does not support enabling Intel SGX device dynamically.

SGX_DISABLED

Intel SGX device is disabled.

SGX_DISABLED_SCI_AVAILABLE

Intel SGX device is disabled, but a Software Control Interface is available to enable it dynamically.

Requirements

Header	`sgx_status.h`

Error Codes

Table 14 Error code

Value	Error Name	Description
0x0000	SGX_SUCCESS	
0x0001	SGX_ERROR_ UNEXPECTED	An unexpected error.
0x0002	SGX_ERROR_ INVALID_ PARAMETER	The parameter is incorrect.
0x0003	SGX_ERROR_ OUT_OF_ MEMORY	There is not enough memory available to complete this operation.
0x0004	SGX_ERROR_ ENCLAVE_ LOST	The enclave is lost after power transition.
0x0005	SGX_ERROR_ INVALID_	The API is invoked in incorrect order or state.

	STATE	
0x0006	SGX_ERROR_ VMM_ INCOMPATIBLE	The virtual machine monitor is not compatible.
0x0007	SGX_ERROR_ HYPERV_ ENABLED	Incompatible versions of Windows* 10 OS and Hyper-V* are detected. In this case, you need to disable Hyper-V on the target machine.
0x0008	SGX_ERROR_ FEATURE_NOT_ SUPPORTED	The feature has been deprecated and is not longer supported.
0x1001	SGX_ERROR_ INVALID_ FUNCTION	The ECALL/OCALL function index is incorrect.
0x1003	SGX_ERROR_ OUT_OF_TCS	The enclave is out of TCS.
0x1006	SGX_ERROR_ ENCLAVE_ CRASHED	The enclave has crashed.
0x1007	SGX_ERROR_ ECALL_NOT_ ALLOWED	ECALL is not allowed at this time. For examples: • ECALL is not public. • ECALL is blocked by the dynamic entry table. • A nested ECALL is not allowed during global initialization.
0x1008	SGX_ERROR_ OCALL_NOT_ ALLOWED	OCALL is not allowed during exception handling.
0x2000	SGX_ERROR_ UNDEFINED_ SYMBOL	The enclave contains an import table.
0x2001	SGX_ERROR_ INVALID_ ENCLAVE	The enclave image is incorrect.
0x2002	SGX_ERROR_ INVALID_ ENCLAVE_ID	The enclave ID is invalid.
0x2003	SGX_ERROR_ INVALID_ SIGNATURE	The signature is invalid.
0x2004	SGX_ERROR_ NDEBUG_	The enclave is signed as product enclave and cannot be created as a debuggable enclave.

	ENCLAVE	
0x2005	SGX_ERROR_ OUT_OF_EPC	There is not enough EPC available to load the enclave or one of the Architecture Enclaves needed to complete the operation requested.
0x2006	SGX_ERROR_ NO_DEVICE	Cannot open device.
0x2007	SGX_ERROR_ MEMORY_MAP_ CONFLICT	Page mapping failed in driver.
0x2009	SGX_EEROR_ INVALID_ METADATA	The metadata is incorrect.
0x200C	SGX_ERROR_ DEVICE_BUSY	Device is busy.
0x200D	SGX_ERROR_ INVALID_ VERSION	Metadata version is inconsistent between uRTS and `sgx_sign` or the uRTS is incompatible with the current platform.
0x200E	SGX_ERROR_ MODE_ INCOMPATIBLE	The target enclave (32/64 bit or HS/Sim) mode is incompatible with the uRTS mode.
0x200F	SGX_ERROR_ ENCLAVE_ FILE_ACCESS	Can't open enclave file.
0x2010	SGX_ERROR_ INVALID_MISC	The MiscSelect/MiscMask settings are incorrect.
0x3001	SGX_ERROR_ MAC_ MISMATCH	Indicates report verification error.
0x3002	SGX_ERROR_ INVALID_ ATTRIBUTE	The enclave is not authorized.
0x3003	SGX_ERROR_ INVALID_ CPUSVN	The CPU SVN is beyond the CPU SVN value of the platform.
0x3004	SGX_ERROR_ INVALID_ ISVSVN	The ISV SVN is greater than the ISV SVN value of the enclave.
0x3005	SGX_ERROR_ INVALID_ KEYNAME	Unsupported key name value.

0x4001	SGX_ERROR_ SERVICE_ UNAVAILABLE	AE service did not respond or the requested service is not supported.
0x4002	SGX_ERROR_ SERVICE_ TIMEOUT	The request to AE service timed out.
0x4003	SGX_ERROR_ AE_INVALID_ EPIDBLOB	Indicates an EPID blob verification error.
0x4004	SGX_ERROR_ SERVICE_ INVALID_ PRIVILEDGE	Enclave has no privilege to get launch token.
0x4005	SGX_ERROR_ EPID_MEMBER_ REVOKED	The EPID group membership has been revoked. The platform is not trusted. Updating the platform and retrying will not remedy the revocation.
0x4006	SGX_ERROR_ UPDATE_NEEDED	Intel® SGX needs to be updated.
0x4007	SGX_ERROR_ NETWORK_ FAILURE	Network connecting or proxy setting issue is encountered.
0x4008	SGX_ERROR_ AE_SESSION_ INVALID	The session is invalid or ended by server.
0x400a	SGX_ERROR_ BUSY	The requested service is temporarily not available.
0x400c	SGX_ERROR_ MC_NOT_ FOUND	The Monotonic Counter does not exist or has been invalidated.
0x400d	SGX_ERROR_ MC_NO_ ACCESS_ RIGHT	The caller does not have the access right to the specified VMC.
0x400e	SGX_ERROR_ MC_USED_UP	No monotonic counter is available.
0x400f	SGX_ERROR_ MC_OVER_ QUOTA	Monotonic counters reached quota limit.
0x5001	SGX_ERROR_ EFI_NOT_ SUPPORTED	The OS does not support EFI.

0x5002	SGX_ERROR_NO_PRIVILEGE	You do not have enough privileges to perform the operation.

Appendix

This topic provides the following reference information:

- Unsupported MSVC* Options for Enclaves
- Unsupported Intel® Compiler Options for Enclaves
- Unsupported Intel® Compiler Libraries
- Unsupported Intrinsics
- Unsupported C Standard Functions
- Unsupported C++ Standard
- Unsupported C and C++ Keywords
- Unsupported Instructions within an Enclave

Unsupported MSVC* Options for Enclaves

The following MSVC* compiler options are not supported to build enclaves:

Table 15 Unsupported MSVC Compiler Options

Option	Description	Remark
`/clr`	Enables applications and components to use features of the common language runtime.	

`/MD`
`/MDd`
`/MT`
`/MTd`
	Selects run-time library.	Linking of DLL's within enclave not allowed. Instead, use Intel® SGX trusted libraries.

`/EHa`	Exception handling model that catches both asynchronous (structured) and synchronous (C++) exceptions.	C++ exceptions are supported inside an enclave, but SEH is not supported.
`/fp`	Specify the floating-point behavior.	Not supported in the Intel® SGX version of the Intel® numeric library.
`/Qimprecise_fwaits`	Removes fwait commands inside try blocks	Not supported in the Intel® SGX version of the Intel numeric library.
`/Qpar` `/Qpar-report`	Enables the compiler's auto-parallelizer feature to automatically parallelize loops in the code.	
`/Fx`	Produces a copy of each source file with injected	

	code merged into the source.	
/GZ	Performs the same operations as the /RTC (Run-Time Error Checks) option.	
/RTC	Used to enable and disable the run-time error checks feature, in conjunction with the runtime_checks pragma.	
/openmp	Causes the compiler to process #pragma omp.	Microsoft* OpenMP* library needs to be linked, which is not self-contained.
/LN	Specifies that an assembly manifest should not be inserted into the output file.	This option is related to common language runtime (/clr) option. As Intel® SGX does not support the /clr option, this option will not be supported.
/analyze	Enable code analysis.	
/hotpatch	Create Hotpatchable Image.	Enclave code cannot be changed after it has been loaded.
/QIPF_B /QIPF_C /QIPF_fr32 /QIPF_noPIC /QIPF_restrict_plabels	Causes compiler not to generate the corresponding instructions.	

Unsupported Intel® Compiler Options for Enclaves

The following Intel® compiler options are not supported to build enclaves:

Table 16 Unsupported Intel® Compiler Options

Option	Description
/hotpatch[:n]	This code generation option is not applicable.
/Qxcode	This option does not apply in Intel® SGX because for this check to be effective, the source file containing the main program or the dynamic library main function should be compiled with this option enabled. Since this

	compiler option does not have the intended behavior (host architecture check), then the /Qax or /arch options are recommended.

`/Qcilk-serialize`
`/Qguide-file[:filename]`
`/Qguide-file-append[:filename]`
`/Qipp[:lib]`
`/Qopt-matmul[-]`
`/Qtbb`
	These advanced optimization options are not supported.

`/Qinstrument-functions[-]`
`/Qprof-data-order[-]`
`/Qprof-dir dir`
`/Qprof-file file`
`/Qprof-func-order[-]`
`/Qprof-gen`
`/Qprof-hotness-threshold:n`
`/Qprof-src-dir[-]`
`/Qprof-src-root:dir`
`/Qprof-src-root-cwd`
`/Qprof-use[:keyword]`
`/Qprof-use-`
`/Qprof-value-profiling[:keyword]`
`/Qprofile-functions`
`/Qprofile-loops:keyword`
`/Qprofile-loops-report[:n]`
`/Qcov-dir:dir`
`/Qcov-file:filename`
`/Qcov-gen[-]`
`/Qfnsplit[-]`
	These profile guided optimizations (PGO) options are not supported.

`/Qtcollect[:lib]`
`/Qtcollect-filter:filename`
`/Qtcheck`
	These optimization report options are not supported.

`/Qopenmp`
`/Qopenmp-stubs`
`/Qopenmp-report[:n]`
`/Qopenmp-lib:type`
`/Qopenmp-task:model`
`/Qopenmp-threadprivate:type`
`/Qpar-affinity:[modifier,...]`
`type[,permute][,offset]`
`/Qpar-num-threads:n`
`/Qpar-report[n]`
`/Qpar-runtime-control[n]`
`/Qpar-runtime-control-`
`/Qpar-schedule-keyword[[:]n]`
`/Qpar-threshold[[:]n]`
`/Qparallel`
`/Qparallel-source-info[:n]`
`/Qparallel-source-info-`
`/Qpar-adjust-stack:n.`
	These OpenMP* and parallel processing options are not supported.

`/Qinline-calloc[-]`	The inlining option is not supported.
`/check:keyword[, keyword...]`	The language option is not supported.

Unsupported Intel® Compiler Libraries

The Intel® compiler libraries that are not supported within an enclave are:

Table 17 Unsupported Intel Compiler Libraries

Option	Description	Remark
`cilkrts.lib`	Cilk runtime system.	
`libchkp.lib` `libchkpwrap.lib`	Run-time pointer checker libraries.	
`libiomp5mt.lib`, `libiompprof5mt.lib`, `libiompstubs5mt.lib`	OpenMP* libraries.	
`libipgo.lib`	Intel® Profile-Guide Optimization (PGO) runtime support library.	
`pdbx.lib`, `pdbxinst.lib`	Intel® Parallel Debugger Extension runtime libraries.	
`libicaio.lib`	Asynchronous I/O library.	I/O is not supported in an enclave.
`libbfp754.lib`	Binary floating-point math library.	It is not utilized by the compiler.
`libmatmul.lib`	Matrix multiplication library.	It depends on the OpenMP library.

Unsupported Intrinsics

The majority of the intrinsics are valid within an enclave. The Microsoft standard instrinsic header file `<intrin.h>` can be included. However, not all the intrinsics that are defined are valid within an enclave. All math and advanced instruction set intrinsics can be used within an enclave. The intrinsics which are NOT valid within an enclave are consistent with instructions that are not supported within an enclave. They generally fall into the following categories:

- I/O related.
- Instructions requiring ring 0 privilege or could change privilege level.
- Operating system or system related functions.
- Intrinsics which are considered non-secure and have secure alternatives.

NOTE

The Intel® SGX SDK has stub implementations, `sgx_intrin.h` for the intrinsics that are not valid within an enclave. These stubs will cause a compiler warning when an unsupported instrinsic is used.

The following intrinsics should not be used within an enclave:

Table 18 Unsupported MSVC Compiler Intrinsics

Not Supported: FS/GS related			
__addgsbyte	__addgsword	__addgsdword	__addgsqword
__incgsbyte	__incgsword	__incgsdword	__incgsqword
__writegsbyte	__writegsword	__writegsdword	__writegsqword
__addfsbyte	__addfsword	__addfsdword	
__incfsbyte	__incfsword	__incfsdword	
__writefsbyte	__writefsword	__writefsdword	__writefsqword
Not Supported: Interrupt/Debug related			
_enable	_disable	__halt	__int2c
Not Supported: I/O related			
_inp, inp	_inpd, inpd	_inpw, inpw	
_out, out	_outp, outd	_outw, outw	
__inbyte	__inword	__indword	
__outbyte	__outword	__outdword	
__inbytestring	__inwordstring	__indwordstring	
__outbytestring	__outwordstring	__outdwordstring	
Not Supported: VMX related			
__vmx_off	__vmx_on	__vmx_vmclear	__vmx_ vmlaunch
__vmx_vmptrld	__vmx_vmptrst	__vmx_vmread	__svm_vmre- sume
__vmx_wmwrite	__nvreg_save_ fence	__nvreg_restore_ fence	
__svm_clgi	__svm_invlpga	__svm_skinit	__svm_stgi
__svm_vmload	__svm_vmrun	__svm_vmsave	
Not Supported: Architectural State related (many require ring 0 privilege)			

__rdtsc	__rdtscp	__readpmc	
__readcr0	__readcr2	__readcr3	__readcr4
__readcr8	__readdr	__writedr	_invpcid
__writecr0	__writecr3	__writecr4	__writecr8
__readeflags	__writeeflags	_setjmp	_setjmpex
__readmsr	__writemsr	__lidt	__sidt
__getcallerseflags	__segmentlimit	__wbinvd	__invlpg
_AddressOfReturnAddress	_ReturnAddress		
Not Supported: Non-secure (use secure alternative)			
_strset	strcat	strcpy	
_wcsset	wcscat	wcscpy	

NOTE:

Even though the CPUID instruction is illegal inside enclaves, the intrinsics `__cpuid` and `__cpuidex` are supported because they are replaced with calls to functions `sgx_cpuid` and `sgx_cpuidex`.

Unsupported C Standard Functions

You cannot use the following Standard C functions within the enclave; otherwise, the compilation would fail.

Table 19 Unsupported C Standard Functions

Header file	Header file in SGX?	Unsupported definition	
		Macros/Types	Functions
complex.h	No	`complex, _complex_I, imaginary, _imaginary_I, I, #pragma STDC CX_LIMITED_RANGE on-off-switch`	`cacos(), cacosf(), cacosl(), casin(), casinf(), casinl(), catan(), catanf(), catanl(), ccos(), ccosf(), ccosl(), csin(), csinf(), csinl(), ctan(), ctanf(), ctanl(), cacosh(), cacoshf(), cacoshl(), casinh(), casinhf(), casinhl(), catanh(), catanhf(), catanhl(), ccosh(), ccoshf(), ccoshl(), csinh(), csinhf(), csinhl(), ctanh(), ctanhf(), ctanhl(), cexp(), cexpf(), cexpl(), clog(), clogf(), clogl(), cabs`

			(), cabsf(), cabsl(), cpow(), cpowf(), cpowl(), csqrt(), csqrtf(), csqrtl(), carg(), cargf(), cargl(), cimag(), cimagf(), cimagl(), conj(), conjf(), conjl(), cproj(), cprojf(), cprojl(), creal(), crealf(), creall()
fenv.h	No	fenv_t, fexcept_t, FE_DIVBYZERO, FE_INEXACT, FE_INVALID, FE_OVERFLOW, FE_UNDERFLOW, FE_ALL_EXCEPT, FE_DOWNWARD, FE_TONEAREST, FE_TOWARDZERO, FE_UPWARD, FE_DFL_ENV, #pragma STDC FENV_ACCESS on-off-switch	feclearexcept, fegetexceptflag, feraiseexcept, fesetexceptflag, fetestexcept, fegetround, fesetround, fegetenv, feholdexcept, fesetenv, feupdateenv
inttypes.h	Yes	SCNdN, SCNiN, SCNoN, SCNuN, SCNxN, SCNdLEASTN, SCNiLEASTN, SCNoLEASTN, SCNuLEASTN, SCNxLEASTN, SCNdFASTN, SCNiFASTN, SCNoFASTN, SCNuFASTN, SCNxFASTN, SCNdMAX, SCNiMAX, SCNoMAX, SCNuMAX, SCNxMAX, SCNdPTR, SCNiPTR, SCNoPTR, SCNuPTR, SCNxPTR,	wcstoimax(), wcstoumax()
locale.h	No	LC_ALL, LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME, struct lconv	setlocale(), localeconv()
setjmp.h	No	jmp_buf	setjmp(), longjmp()
signal.h	No	sig_atomic_t, SIG_DFL, SIG_ERR, SIG_IGN, SIGABRT,	signal(), raise()

		SIGFPE, SIGILL, SIGINT, SIGSEGV, SIGTERM,	
stdio.h	Yes	fpos_t, _IOFBF, _IOLBF, _IONBF, FILENAME_MAX, FOPEN_MAX, L_tmpnam, SEEK_CUR, SEEK_END, SEEK_SET, TMP_MAX, stderr, stdin, stdout,	remove(), rename(), tmpfile(), tmpnam(), fclose(), fflush(), fopen(), freopen(), setbuf(), setvbuf(), fprintf(), fscanf(), printf(), scanf(), sprintf(), sscanf(), vfprintf(), vfscanf(), vprintf(), vscanf(), vsprintf(), vsscanf(), fgetc(), fgets(), fputc(), fputs(), getc(), getchar(), gets(), putc(), putchar(), puts(), ungetc(), fread(), fwrite(), fgetpos(), fseek(), fsetpos(), ftell(), rewind(), clearerr(), feof(), ferror(), perror()
stdlib.h	Yes		rand(), srand(), atexit(), exit(), _Exit(), getenv(), system()
string.h	Yes		strcpy(), strcat(), strstr()*
tgmath.h	No		
time.h	Yes		clock(), mktime(), time(), ctime(), gmtime(), localtime()
wchar.h	Yes		fwprintf(), fwscanf(), swscanf(), vfwprintf(), vfwscanf(), vswscanf(), vwprintf(), vwscanf(), wprintf(), wscanf(), fgetwc(), fgetws(), fputwc(), fputws(), fwide(), getwc(), getwchar(), putwc(), putwchar(), ungetwc(), wcstod(), wcstof(), wcstold(), wcstol(), wcstoll(), wcstoul(), wcstoull(), wcscpy(), wcscat(), wcsftime(),wctob()
wctype.h	Yes		iswalnum(), iswalpha(), iswblank(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(), wctype(), towlower(), towupper(), towctrans(), wctrans()

(*) The trusted standard C library does not support `char strstr(const char*, const char*)`. The trusted standard C library support sthe variant `const char* strstr (const char*, const char*)`.

In addition to the 'C' standard `memset()` function, the trusted 'C' library (`sgx_tstdc`) also supports `memset_s()`. The following is a note on the recommended use of `memset()` versus `memset_s()`.

NOTE

Trusted C library is enhanced to avoid format string attacks. Any attempts to use `%n` in printf-family functions such as `snprintf` will result in a run-time error.

It is appropriate to use `memset()` to initialize buffers and clear buffers that do not contain secret data. If the purpose is to clear a buffer that contained secret data before deletion of the that buffer, you should not use the `memset()` function and should use the `memset_s()` function instead. The problem with using `memset()` in this scenario is that the compiler can optimize out the write to memory to clear the buffer so that it will not be performed (the compiler does this since it recognizes the subsequent deletion of the buffer). The use of `memset_s()` guarantees the compiler will *not* optimize away the write to memory and thus ensuring the secret data is cleared. However, it is not recommended that `memset_s()` should always be used in place of `memset()` since the implementation of `memset_s()` is not performance optimized.

Unsupported C++ Standard

The following table lists unsupported C++03 classes and functions inside the enclave. Also, the table does not include unsupported C functions. See Unsupported C Standard Functions for detailed information.

Table 20 Unsupported C++ Standard Classes and Functions

Class Category	Partially Supported	Unsupported Classes
Stream Iterators	No	istream_iterator, ostream_iterator, istreambuf_iterator, ostreambuf_iterator
Input/Output Library	No	basic_streambuf, basic_istream, basic_ostream, basic_iostream, basic_filebuf, basic_ifstream, basic_ofstream, basic_fstream, basic_stringbuf, basic_istringstream, basic_ostringstream, basic_stringstream
Locales	No	locale, use_facet, has_facet

Unsupported C and C++ Keywords

The following keywords are not supported in an enclave:

Table 21 Unsupported C and C++ Keywords

Category	Unsupported Keywords
Structured Exception Handling	__try, __except, __finally, __leave
Managed Extension	__abstract, __box, __sealed, __value, __delegate, __gc, __ nogc, __property, __try_cast, __pin
Common Language Runtime	__identifier, nullptr, array, value class, delegate, enum class, typeid, enum struct, generic, event, finally, initonly, 'for each, in', ref struct, friend_as, gcnew, safecast, interface class, interface struct, ref class, interior_ptr, literal, value struct, property
Timing and Event	__event, __hook, __unhook, __raise, event
Additional Keywords	dllimport, __unaligned, __w64

Unsupported Instructions within an Enclave

Some CPU instructions cannot be used within an enclave. Using them will either cause a #UD (undefined instruction) or #GP (general-protection fault).

Table 22 Unsupported Instructions in SGX

Type	Instructions
VMEXIT generating instructions are not allowed because a VMM cannot update the enclave. Generates a #UD.	CPUID, GETSEC, RDPMC, RDTSC, RDTSCP, SGDT, SIDT, SLDT, STR, VMCALL, VMFUNC
I/O instructions (also VMEXIT). Generates #UD.	IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD
Instructions which access segment registers will also generate #UD.	Far CALL, Far JMP, Far RET, INT n/INTO, IRET, LDS/LES/LFS/LGS/LSS, MOV to DS/ES/SS/FS/GS, POP DS/ES/SS/FS/GS, SYSCALL, SYSENTER
Instructions that try to reenter the enclave. Generates #GP.	ENCLU[EENTER], ENCLU [ERESUME]