



# Intel<sup>®</sup> Software Guard Extensions (Intel<sup>®</sup> SGX) SDK for Windows<sup>\*</sup> OS

**Developer Reference**

---

## Legal Information

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel technologies features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at Intel.com, or from the OEM or retailer.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting [www.intel.com/design/literature.htm](http://www.intel.com/design/literature.htm).

Intel, the Intel logo, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.

### 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

\* Other names and brands may be claimed as the property of others.

© Intel Corporation.

This software and the related documents are Intel copyrighted materials, and your use of them is governed by the express license under which they were provided to you (**License**). Unless the License provides otherwise, you may not use, modify, copy, publish, distribute, disclose or transmit this software or the related documents without Intel's prior written permission.

This software and the related documents are provided as is, with no express or implied warranties, other than those that are expressly stated in the License.

## Revision History

Revision Number	Description	Revision Date
1.1	Intel® SGX Win 1.1 release	September 2015
1.6	Intel® SGX Win 1.6 release	April 2016
1.7	Intel® SGX Win 1.7 release	November 2016
1.8	Intel® SGX Win 1.8 release	June 2017
1.9	Intel® SGX Win 1.9 release	October 2017
1.9.5	Intel® SGX Win 1.9.5 release	January 2018
1.9.6	Intel® SGX Win 1.9.6 release	March 2018
2.0.0	Intel® SGX Win 2.0.0 release	April 2018
2.0.1	Intel® SGX Win 2.0.1 release	April 2018
2.1	Intel® SGX Win 2.1 release	August 2018
2.2	Intel® SGX Win 2.2 release	November 2018
2.2.3	Intel® SGX Win 2.2.3 release	February 2019
2.3	Intel® SGX Win 2.3 release	March 2019
2.4	Intel® SGX Win 2.4 release	June 2019
2.5	Intel® SGX Win 2.5 release	October 2019
2.5.1	Intel® SGX Win 2.5.1 release	November 2019
2.6	Intel® SGX Win 2.6 release	January 2020
2.7	Intel® SGX Win 2.7 release	March 2020
2.7.1	Intel® SGX Win 2.7.1 release	April 2020
2.8	Intel® SGX Win 2.8 release	June 2020
2.9	Intel® SGX Win 2.9 release	August 2020
2.10	Intel® SGX Win 2.10 release	September 2020
2.11	Intel® SGX Win 2.11 release	November 2020
2.12	Intel® SGX Win 2.12 release	January 2021
2.13	Intel® SGX Win 2.13 release	June 2021
2.14	Intel® SGX Win 2.14 release	September 2021
2.14.1	Intel® SGX Win 2.14.1 release	November 2021
2.15	Intel® SGX Win 2.15 release	March 2022
2.16	Intel® SGX Win 2.16 release	June 2022
2.17	Intel® SGX Win 2.17 release	November 2022
2.18	Intel® SGX Win 2.18 release	March 2023
2.19	Intel® SGX Win 2.19 release	July 2023

## Introduction

Intel provides the Intel® Software Guard Extensions (Intel® SGX) SDK Developer Reference for software developers who wish to harden their application security using the Intel Software Guard Extensions technology.

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

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

---

### **NOTE**

Intel® Software Guard Extensions (Intel® SGX) technology is only available on the 6th Generation Intel® Core(TM) Processor or newer.

---

## 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 Microsoft\* 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 debugger to view its content. 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.<sup>1</sup>

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.

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 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 SDK.

## Terminology and Acronyms

AE	Architectural enclaves. Enclaves that are part of the Intel® Software Guard Extensions framework. They include the quoting
----	--

---

<sup>1</sup>From an enclave standpoint, the operating system and VMM are not trusted components, either.

	enclave (QE), provisioning enclave (PvE), launch enclave (LE), provisioning certification enclave (PCE) and the platform service enclave (PSE).
Attestation	Prove authenticity. In case of platform attestation, prove the identity of the platform.
CA	Certificate Authority.
ECALL	Enclave call. A function call that enters the enclave.
ECF	Enclave Configuration File.
ECDH	Elliptic curve Diffie–Hellman.
EDL	Enclave Definition Language.
Intel® EPID	Intel® Enhanced Privacy ID.
FIPS	Federal Information Processing Standards developed by NIST for use in computer systems government-wide.
FIPS 140-2	Standard that defines security requirements for cryptographic modules and is required for sales to the Federal Governments.
HSM	Hardware Security Module.
Attestation Service	Attestation Service for Intel® Software Guard Extensions.
ISV	Independent Software Vendor.
KE	Key Exchange.
LE	Launch enclave, an architectural enclave from Intel, involved in the licensing service.
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.
PCE	Provisioning certification enclave, an architectural enclave from Intel, involved in the Intel® Enhanced Privacy ID (Intel® EPID) Provision service to provide provisioning certificate.
PSE	Platform service enclaves, architectural enclaves from Intel. Including PSE-pr (long-term paring) and PSE-Op (session management).
Intel® SGX PSW	Platform Software for Intel® Software Guard Extensions.
PvE	Provisioning enclave, an architectural enclave from Intel, involved in the Intel® Enhanced Privacy ID (Intel® EPID) Provision service to handle the provisioning protocol.
QE	Quoting enclave, an architectural enclave from Intel, involved in

	the quoting service.
Intel® SGX	Intel® Software Guard Extensions.
SigRL	Signature revocation list
SMK	Session MAC key.
SP	Service Provider.
SVN	Security version number. Used to version security levels of both hardware and software components of the Intel® 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.
TCS	Thread Control Structure.
TLS	Thread Local Storage.
TLS	Transport Layer Security.
tRTS	Trusted Run Time System
uRTS	Untrusted Run Time System
Intel® SGX SSL	Intel® Software Guard Extensions SSL cryptographic library based on the OpenSSL. Provides cryptographic services for Intel® Software Guard Extensions enclave applications.

## Setting up an Intel® Software Guard Extensions Project

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

- [Using Microsoft\\* Visual Studio\\* Intel® Software Guard Extensions Wizard](#)
- [Using Microsoft\\* Visual Studio\\* Intel® Software Guard Extensions Add-in](#)
- [Enclave Project Files](#)
- [Microsoft\\* Visual Studio\\* Project Settings](#)

---

### **NOTE**

The Visual Studio 2019 Add-in Tool is designed to work with the Visual Studio 2019 Professional Environment. While Visual Studio 2019 Professional is the recommended environment, the tools may also be installed with the Community and Enterprise versions of Visual Studio 2019.

---

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

Intel® Software Guard Extensions 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 SDK.

#### **Creating an Enclave**

1. On the menu bar of Microsoft\* Visual Studio\*, choose **File-->New-->Project**.

The **New Project** dialog box opens.

2. Select **Intel® SGX Enclave Project**, then click **Next**. Enter name, location, and solution name in the appropriate fields like any other Microsoft\* Visual Studio\* project.

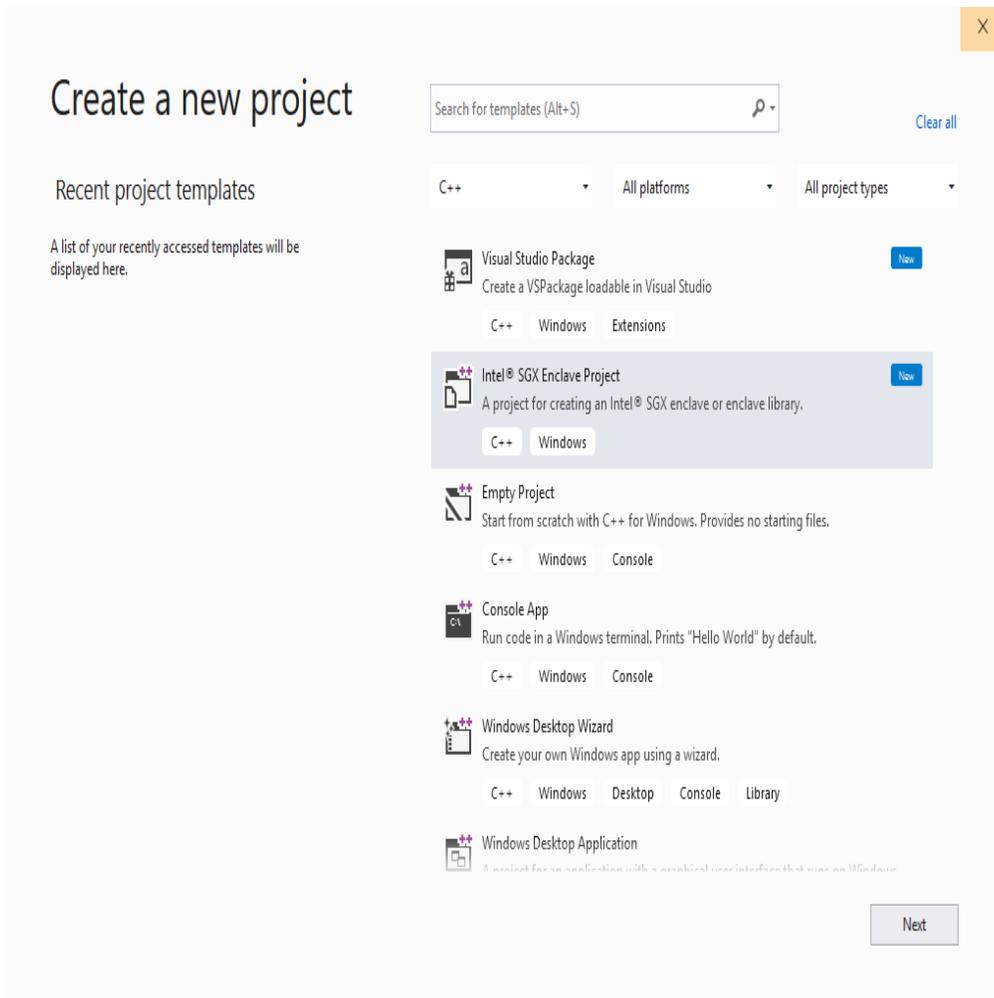


Figure 1 Intel® SGX Wizard: New Project Creation

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

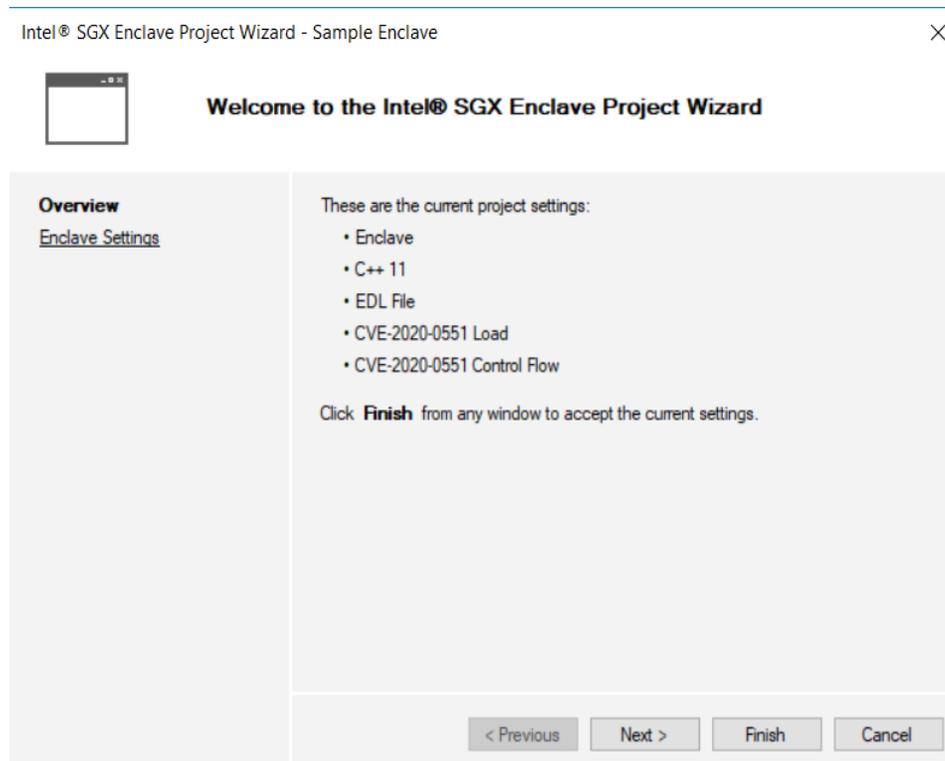


Figure 2 Intel® SGX Wizard: Welcome Dialog

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

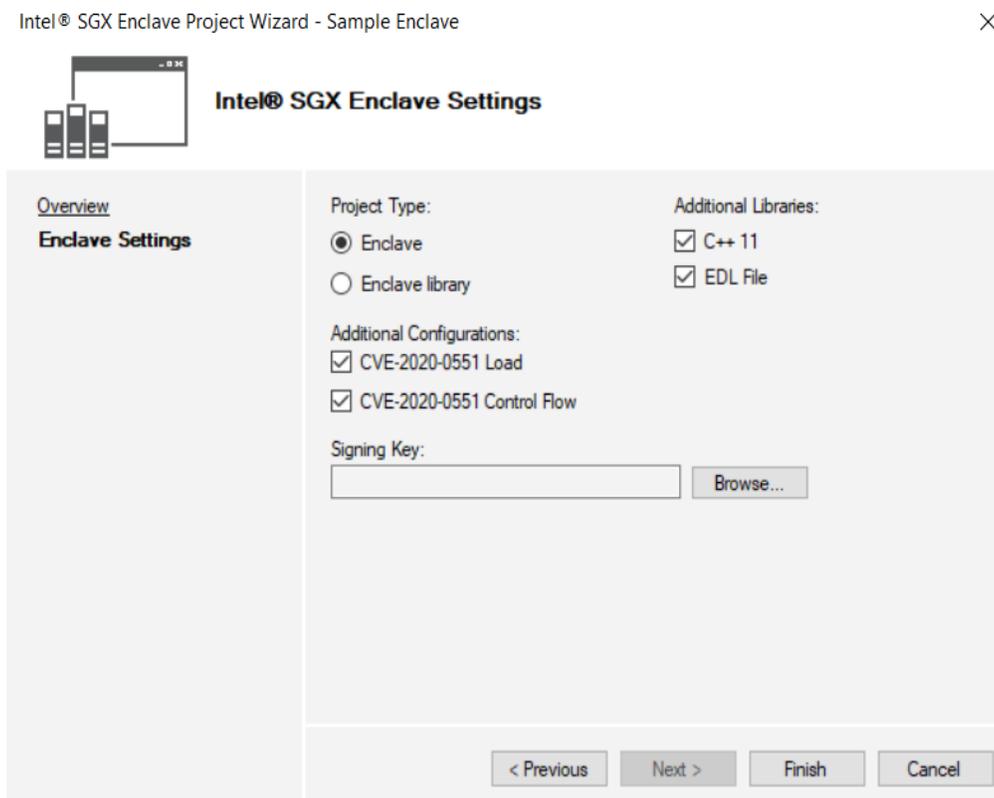


Figure 3 Intel® SGX Wizard: Enclave Settings

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++ 11 – Link C++ 11 with the enclave project.
    - EDL File - Create an EDL file in 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](#)).

- Addition Configurations:
  - CVE-2020-0551 Load – Add CVE-2020-0551 mitigation Load level configuration.
  - CVE-2020-0551 Control Flow – Add CVE-2020-0551 mitigation Control Flow level configuration.

---

**NOTE:**

See [Enable CVE-2020-0551 Mitigation](#) for details on these new configurations.

---

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

---

**NOTE:**

The Wizard requires Visual C++ tools installed, which is no longer done by default in Visual Studio 2019. Please make sure the C++ tools are installed before using the Wizard to create an enclave project.

---

---

**NOTE:**

The Wizard creates an enclave project with several files. See [Enclave Project Files](#) for a detailed file list.

---

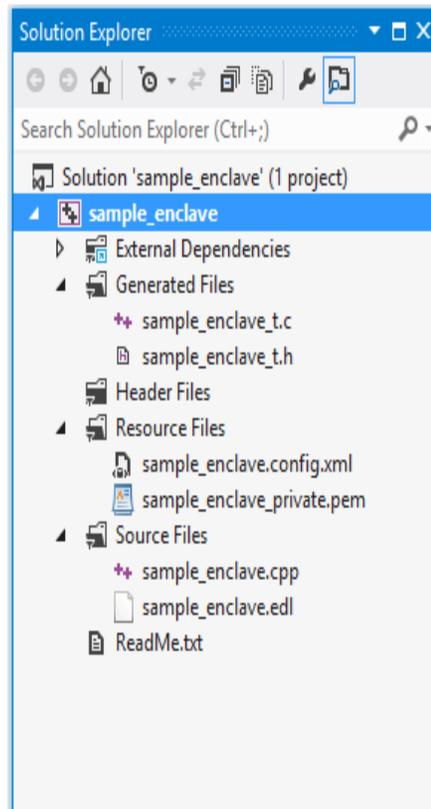


Figure 4 Intel® SGX Wizard: Solution Explorer

### 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 Project Configuration](#) helps to add [CVE-2020-0551 mitigation](#) enabled configurations into exist trust projects.

### 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 you the option to update the following enclave settings:

- ProdID
- ISVSVN
- StackMaxSize
- HeapMaxSize
- TCSNum
- TCSPolicy
- DisableDebug
- MiscSelect
- MiscMask
- EnableKSS
- ISVEXTPRODID\_H
- ISVEXTPRODID\_L
- ISVFAMILYID\_H
- ISVFAMILYID\_L

### 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.

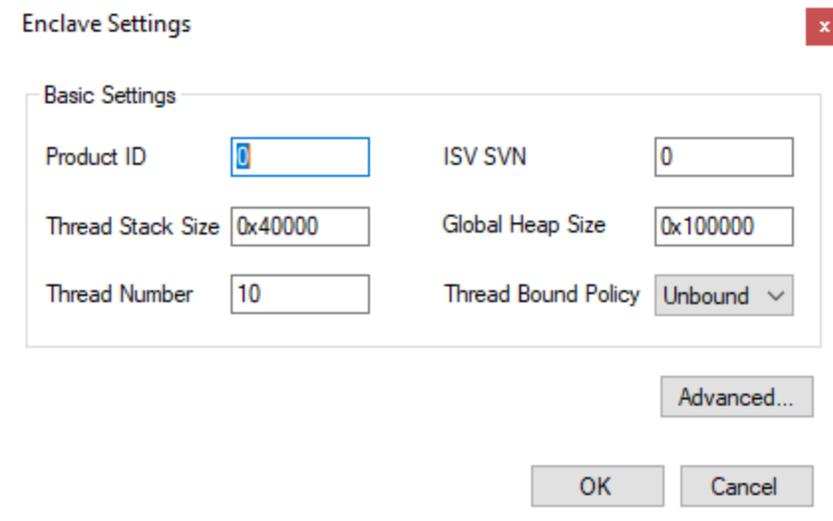


Figure 5 Intel® SGX Configuration: Enclave Settings

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:

Advanced Settings

Changing the following settings for current enclave might affect the enclave launch process.

**Debug Selection**

The enclave cannot be debugged

**Misc Settings**

Set the corresponding bits for the Misc Select and Misc Mask.

Misc Select  Misc Mask

Enable KSS

ISV Ext Prod ID:

High 8 Bytes:

Low 8 Bytes:

ISV Family ID:

High 8 Bytes:

Low 8 Bytes:

OK Cancel

Figure 6 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 can be set for Misc Select by default. The recommendation is keeping the default settings.

Check the Enable KSS option if you want to enable Key Separation & Sharing (KSS) for the enclave. The setting corresponds to the element `EnableKSS` of the [Enclave Configuration File](#). You can also set the value for the ISV assigned Extended Product ID and the ISV assigned Family ID from the text box. These settings correspond to the elements `ISVEXTPRODID_H`, `ISVEXTPRODID_L`, `ISVFAMILYID_H`, and `ISVFAMILYID_L` of the [Enclave Configuration File](#) respectively. These elements are relevant to a more flexible model of KSS. Note that you need to enable KSS before setting the ISV assigned Extended Product ID and the ISV assigned Family ID.

### Enclave Preferred Load Address

You may use the linker options `/BASE` and `/FIXED` to build an enclave. `/FIXED` tells the uRTS and the driver to load the enclave at the preferred load address only, which can be specified with the option `/BASE`.

When you use `/BASE`, the enclave PE's `OptionalHeader.ImageBase` is set with the preferred load address. When you specify `/FIXED`, the linker does not generate a relocation section for the enclave and the enclave PE's `Header.Characteristics` will have `IMAGE_FILE_RELOCS_STRIPPED` set. Additionally the PE's `OptionalHeader.DLLCharacteristics` will not have `IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE` set, to indicate the enclave DLL cannot be relocated at load time.

When you sign the enclave, if `sgx_sign.exe` detects that an enclave has `IMAGE_FILE_RELOCS_STRIPPED` set in the enclave PE's `Header.Characteristics`, it will verify that the enclave does not have a relocation section and that `IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE` is not set in the PE `OptionalHeader.DLLCharacteristics`. In addition, `sgx_sign.exe` ensures that the selected `BASE` naturally aligns with the enclave size. If any of these three requirements fails, `sgx_sign.exe` reports an error. The loader `sgx_urts.dll` only attempts to load an enclave after the enclave passes the same three tests. You may still receive an error from `sgx_urts.dll` if the enclave cannot be loaded at the preferred load address.

### Enclave Signing

With the enclave launch control, the enclave signing key for release mode must be stored in secure manner, such as a platform managed by an HSM. All

the release mode enclaves should use two-step signing mechanism. [Enclave Signer Usage 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.

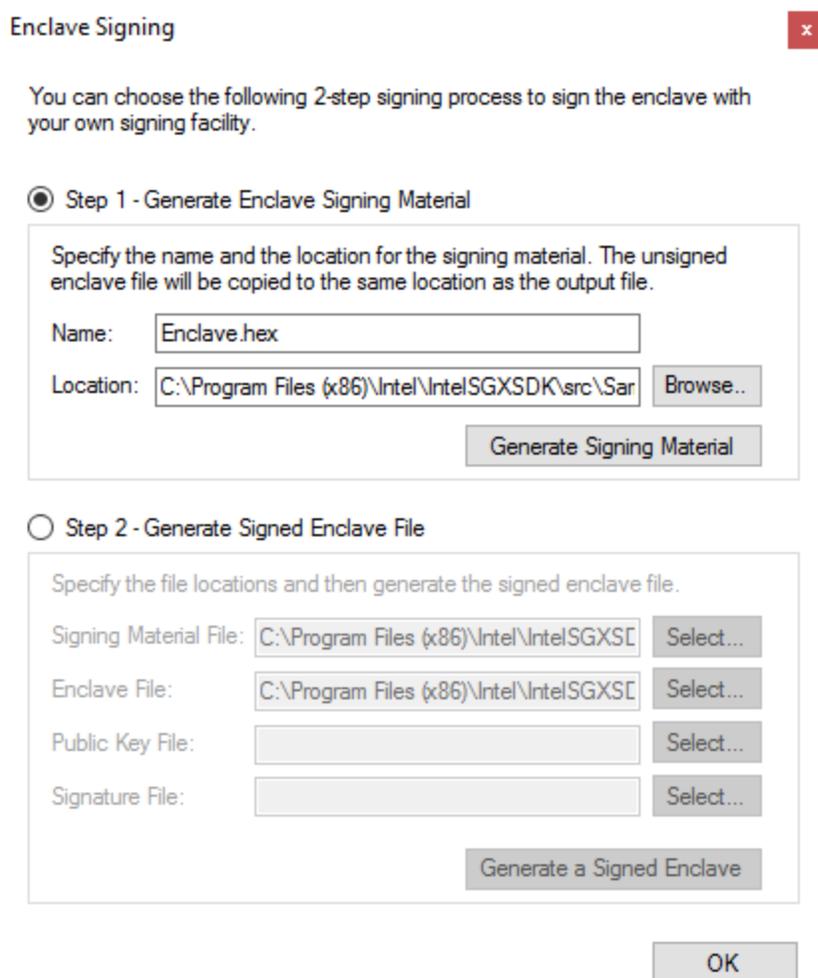


Figure 7 Intel® SGX Configuration: Generate Enclave Signing Material

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, then bring 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**.

## Enclave Signing



You can choose the following 2-step signing process to sign the enclave with your own signing facility.

 Step 1 - Generate Enclave Signing Material

Specify the name and the location for the signing material. The unsigned enclave file will be copied to the same location as the output file.

Name:

Location:

 Step 2 - Generate Signed Enclave File

Specify the file locations and then generate the signed enclave file.

Signing Material File:

Enclave File:

Public Key File:

Signature File:

Figure 8 Intel® SGX Configuration: Generate a Signed Enclave File

The default location for the signing material and the enclave file are specified. Check whether the specified paths 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 of 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.

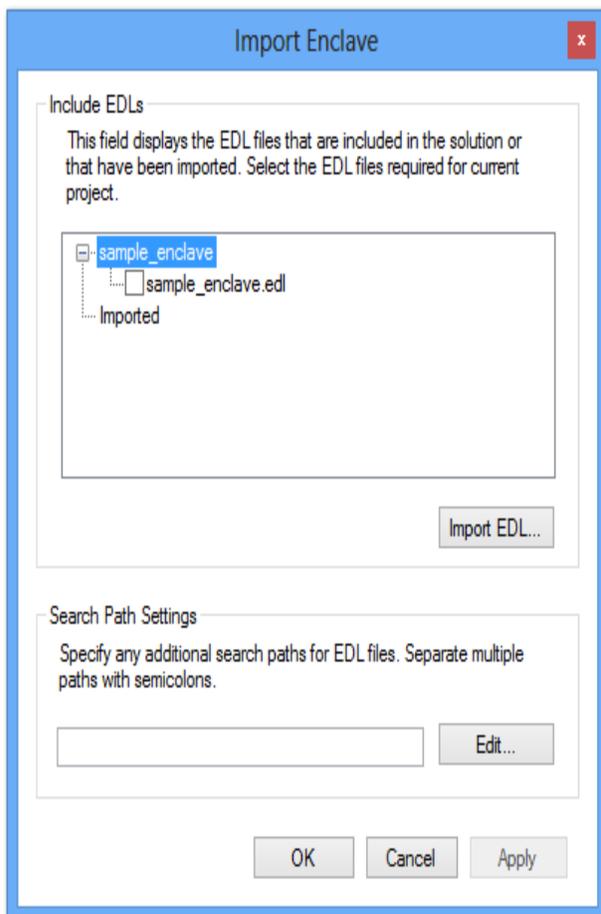


Figure 9 Intel® SGX Configuration: Import Enclave

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.

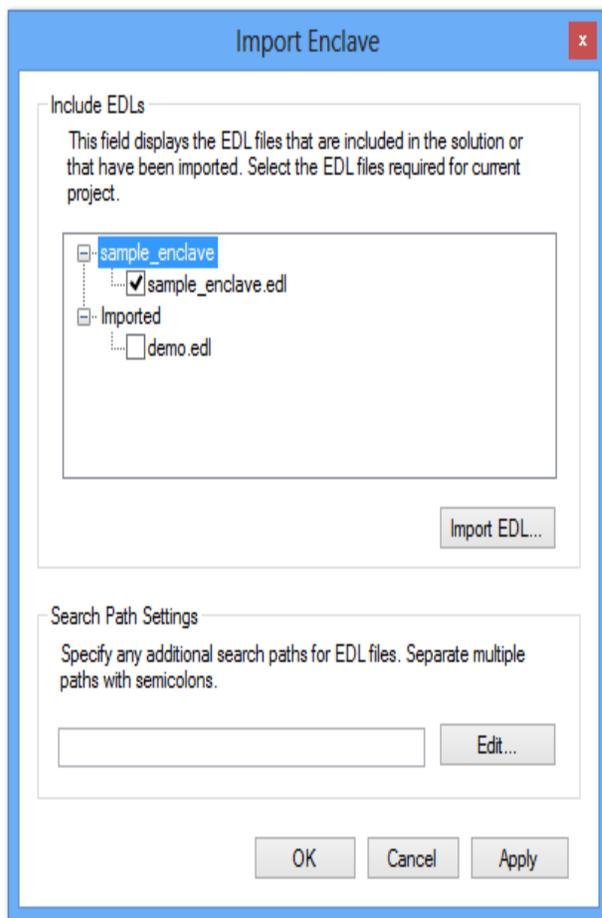


Figure 10 Intel® SGX Configuration: Import EDL File

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**.

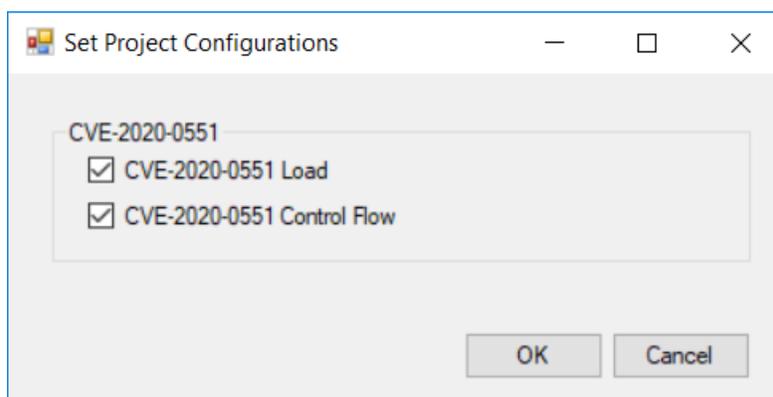
To remove an imported enclave from the untrusted application, unselect the corresponding EDL file and click **OK**. Then the corresponding settings in the untrusted application are removed.

### Enclave Project Configuration

**Enclave Project Configuration** helps you add new configurations into exist trust projects.

To add new configurations:

Open the solution that contains the trust project. Right click the trust project. Select **Intel® SGX Configuration -> Enclave Project Configuration**. A dialog will be shown.



The **CVE-2020-0551** dialog box provides users the interface to select the two levels of CVE-2020-0551 mitigation enabled configurations.

---

#### **NOTE:**

If target trusted project has included corresponding CVE-2020-0551\* configurations already, the tool will do nothing.

---

#### **NOTE:**

See [Enable CVE-2020-0551 Mitigation](#) for details.

---

### 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 [Enclave Signing Tool](#) for a detailed description.

---

## Microsoft\* Visual Studio\* 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:

```
[Intel SGX SDK Install Path]include;[Intel SGX SDK Install Path]include\tlibc;  
[Intel SGX SDK Install Path]include\stlport;
```

#### C/C++->Command Line->Additional Options:

```
/d2FH4-
```

---

#### **NOTE:**

The existing projects should also add /d2FH4- flag to their project settings if they upgrade from VS2017 to VS2019.

---

#### Linker->General->Additional Library Directories:

```
[Intel SGX SDK Install Path]bin\$(Platform)\$(Configuration)
```

#### Linker->Input->Additional Dependencies:

```
HW Configuration: sgx_trts.lib; sgx_tservice.lib; sgx_  
tstdc.lib; sgx_tlibcxx.lib; sgx_tcrypto.lib
```

```
Simulation Configuration: sgx_trts_sim.lib; sgx_tservice_  
sim.lib; sgx_tstdc.lib; sgx_tlibcxx.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 (Win32):

```
"[Intel SGX SDK Install Path]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 "$(Pro-  
jectDir)sample_enclave.config.xml"
```

### **Build Events->Post-Build Event->Command Line (x64):**

```
"[Intel SGX SDK Install Path]bin\x64\release\sgx_
sign.exe" sign -key "$(ProjectDir)sample_enclave_
private.pem" -enclave "$(OutDir)sample_enclave.dll" -out
"$(OutDir)sample_enclave.signed.dll" -config "$(Pro-
jectDir)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](#) 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 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 SDK Tools

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

- **Edger8r Tool**  
Generates interfaces between the untrusted components and enclaves.
- **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.

### Edger8r Tool

The Edger8r tool ships as part of the Intel® Software Guard Extensions SDK. It generates edge routines by reading a user-provided Enclave Description Language (EDL) file. These edge routines define the interface between the untrusted application and the enclave. Normally, the tool runs 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 by default generates four files:

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

- `demo_u.c` – Contains function definitions for untrusted proxies and bridges.

Here is the command line description for the Edger8r tool:

#### Syntax:

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

#### Arguments:

[Options]	Descriptions
<code>--use-prefix</code>	Prefix the untrusted proxy with the enclave name.
<code>--header-only</code>	Generate header files only.
<code>--search-path</code> <path>	Specify the search path of EDL files.
<code>--untrusted</code>	Generate untrusted proxy and bridge routines only.
<code>--trusted</code>	Generate trusted proxy and bridge routines only.
<code>--untrusted-dir</code> <dir>	Specify the directory for saving the untrusted code.
<code>--trusted-dir</code> <dir>	Specify the directory for saving the trusted code.
<code>--preprocessor</code>	Specify the path to the compiler, which is used to pre-process the EDL file.
<code>--help</code>	Print help message showing the command line and options.

If neither `--untrusted` nor `--trusted` is specified, the Edger8r generates 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.

---

## Enclave Signing Tool

The Intel® Software Guard Extensions (Intel® SGX) 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 (see [Enclave Signature Structure](#) below). 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 you 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. In addition, the signing tool can also be used to report metadata information for a signed enclave and to generate the SIGStruct file needed to add the enclave signer to the allowlist.

Table 2 Enclave Signature Structure

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

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

### 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 Signer Usage Examples](#) for more information.

**Table 3 Signing Tool Commands**

Command	Description	Arguments
<code>sign</code>	Sign the enclave using the private key in one step.	Required: -enclave, -key, -out  Optional: -config, -dumpfile, -cssfile
<code>gendata</code>	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 signature structure (see	Required: -enclave, -out  Optional: -config

	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 <code>gendata</code> 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, -unsigned  Optional: -config, -dumpfile, -cssfile
dump	Get the metadata information for a signed enclave file and dump the metadata to a file specified with the <code>-dumpfile</code> option and the SIGSTRUCT to the file specified by the <code>-cssfile</code> option.	Required: -enclave, -dumpfile  Optional: -cssfile

All the valid command options are listed below:

**Table 4 Signing Tool Arguments**

Arguments	Descriptions	
-enclave <file>	Specify the enclave file to be signed or already signed. It is a required argument for the four commands.	
-config <file>	Specify the enclave configuration file It is an invalid argument for the <code>dump</code> command and an optional argument for the other three commands.	
-out <file>	Specify the output file. It is required for the following three commands.	
	<b>Command</b>	<b>Description</b>
	sign	The signed enclave file.
	gendata	The file with the enclave signing material.
catsig	The signed enclave file.	
-key <file>	Specify the signing key file. See <a href="#">File Formats</a> for detailed description.	

	Command	Description
	sign	Private key.
	gendata	Not applicable.
	catsig	Public key.
-sig <file>		Specify the file containing the signature corresponding to the enclave signing material.  Only valid for catsig command.
-unsigned <file>		Specify the file containing the enclave signing material generated by gendata.  Only valid for the catsig command.
-dumpfile		Specify a file to dump metadata information.  It is a required argument for the dump command and an optional argument for sign and catsig
-cssfile		Specify a file to dump the SIGSTRUCT information.  It is an optional argument for the sign, catsig and dump commands.
-resign		By default, sgx_sign reports an error if an input enclave has already been signed. You can force sgx_sign to resign the enclave by providing this option
-version		Print the version information and exit.
-help		Print the help information and exit.

The arguments, including options and filenames, 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.

## 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 enclave signing key added to the allowlist must be managed in a hardware security module. Thus, the ISV's test private key stored in the build platform will not be added to the allowlist 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	A standard DLL.
Signed enclave file	DLL	<code>sgx_sign</code> generates the signed enclave file , which includes the signature, to the enclave file.
Configuration file	XML	See <a href="#">Enclave Configuration File</a> .
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	A dump file of the enclave signing material data to be signed with the private RSA key.
Signature file	RAW	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.
Metadata file	RAW	A dump file containing the SIGStruct metadata for the signed enclave. This file is submitted when there is a request for Intel to add a production enclave to the allowlist.

### Signing Key Files

The enclave signing tool only accepts key files in the PEM format and that are unencrypted. When an enclave project is created for the first time, you have to choose either 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 Signer Usage 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_sig.dat
```

2. At the signing facility, sign the file containing the enclave signing material (`enclave_sig.dat`) and take the resulting signature file (`signature.dat`) 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.dat -unsigned enclave_sig.dat
```

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

The following example illustrates generating a report of metadata information for a signed enclave:

```
sgx_sign dump -enclave enclave_signed.dll -dumpfile metadata_info.txt
```

Intel® SGX SDK provides a tool for operating with both 32-bit and 64-bit enclaves.

## 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.dat -sign private_key.pem -keyform PEM enclave_sig.dat
```

## Enclave Debugger

Only a debug mode enclave can be debugged.

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**.

### Starting and Debugging an Enclaved Application from within Microsoft\* Visual Studio

The native Microsoft\* Visual Studio\* 2019 debugger can debug an enclave, so you can perform normal application debugging such as setting breakpoints or stepping into an enclave.

In earlier versions of Microsoft\* Visual Studio\*, a debug mode enclave can be debugged with the Intel® SGX debugger.

### Attaching to and Debugging an Enclave inside a Running Process

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

In Microsoft\* Visual Studio\*, select the process that you would like to attach to and debug, then click the **Attach** button.

When performing a remote debugging, the host machine needs the Intel® SGX SDK installed; the target machine needs the Microsoft\* Visual Studio\* remote debugger server:

1. On the target machine, launch the Remote Debugging Monitor (`msvs-mon.exe`) that was installed as a part of Microsoft\* Visual Studio\*.
2. On the host machine, set the qualifier as the target machine name or IP address.

---

**NOTE:**

The remote machine may have multiple versions of the Remote Debugging Monitor. Ensure that the version of the Remote Debugging Monitor being run corresponds to the version of Microsoft\* Visual Studio\* that was configured during the installation of the Intel® SGX SDK.

---

See Remote Debugging Setup at <https://docs.microsoft.com/en-us/visualstudio/debugger/remote-debugging>

The 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 parts of the Intel® Software Guard Extensions 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.

### **Attaching to and Debugging an Enclave inside a Running Universal Windows Application**

The enclave inside a Universal Windows Platform (UWP) Application can only be debugged by the attach mode. Please see [Attaching to and Debugging an Enclave inside a Running Process](#) for more details.

### **Enclave Memory Measurement Tool**

An enclave is an isolated environment. The Intel® Software Guard Extensions 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 functions:

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

The tool reports the size of the memory usage in KB. Once you get accurate memory 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-
e=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 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 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:

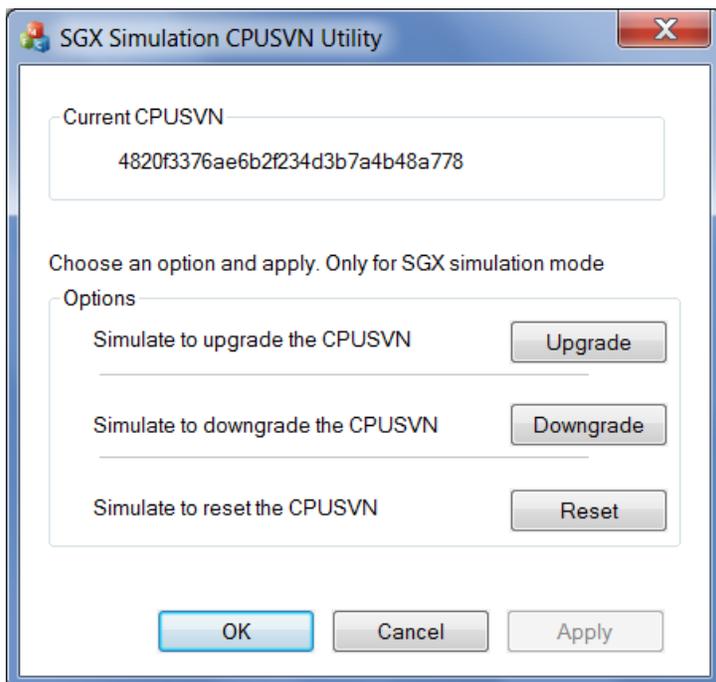
```
sgx_config_cpusvn [Command]
```

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 11 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](#)
- [Loading and Unloading 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, [Edger8r Tool](#) generates trusted and untrusted proxy/bridge functions. [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 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. To check a list of illegal instructions inside an enclave, see [Intel® Software Guard Extensions Programming Reference](#).
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 SDK and they can be used instead.
OS API calls (for example, 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 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 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. Spinlocks, trusted mutex and condition variables API can be used for thread synchronization inside the enclave.
Thread Local Storage (TLS)	Partially	Only implicitly via declspec(thread), Thread_local, or thread_local. 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 SDK provides an API to allow you to register functions, or exception handlers, to handle a limited set of hardware exceptions. See <a href="#">Custom Exception Handling</a> for more details.

### 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 {
```

```

// Add your definition of "secret_t" here
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 during 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 (called by the application):

```

sgx_status_t get_secret(sgx_enclave_id_t eid, secret_t*
secret);

```

Trusted proxy function (called by the enclave):

```

sgx_status_t dump_secret(const secret_t* secret);

```

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 some cases, the code within the enclave needs to call external functions which reside in untrusted (unprotected) memory to use operating system capabilities outside the enclave such as system calls, I/O operations, and so on. This type of function call is named 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 Windows\*OS loads DLLs. The function address 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 debugger.

---

#### **CAUTION:**

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 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 [Importing EDL Libraries](#) 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 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()` that are needed in `sethread_mutex.obj`, which is part of `sgx_tstdc.lib`.

```
sgx_tstdc.lib(sethread_mutex.obj) : error LNK2001: unre-  
solved external symbol _sgx_thread_wait_untrusted_event_  
ocall
```

```
sgx_tstdc.lib(sethread_mutex.obj) : error LNK2001: unre-  
solved external symbol _sgx_thread_set_untrusted_event_  
ocall
```

---

**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.

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

them 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

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

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

Step 2 (optional but highly recommended) – write a trusted, user-friendly wrapper. This function is part of the enclave's trusted code.

The wrapper function `ocall_foo` function will look like:

```
// enclave's trusted code
#include "foo_t.h"
void ocall_foo(int param)
{
    // it is necessary to check the return value of foo()
    if (foo(param) != SGX_SUCCESS)
        abort();
}
```

Step 3 – write an untrusted `foo` function.

```
// untrusted code
void foo(int param)
{
    // the implementation of foo
}
```

The `sgx_edger8r` will generate an untrusted bridge function which will call the 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 [Importing EDL Libraries](#).

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 it 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 SDK, the enclave EDL file must be parsed with the `--use-prefix` option to `sgx_edger8r`. 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 an enclave must have an empty IAT (Import Address Table).

---

### Static Libraries

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

The Intel® Software Guard Extensions Software Development Kit (Intel® SGX SDK) provides the following collection of trusted libraries.

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

<b>Name</b>	<b>Description</b>	<b>Comment</b>
sgx_trts.lib	Intel® SGX Runtime library	Must link when running in HW mode
sgx_trts_sim.lib	Intel® SGX Runtime library (simulation mode)	Must link when running in simulation mode
sgx_tstdc.lib	Standard C library (math, string, and so on.)	Must link
sgx_tcxx.lib	Standard C++ libraries, STL	Optional
sgx_tservice.lib	Data seal/unseal (encryption), trusted Architectural Enclaves support, Elliptic Curve Diffie-Hellman (EC DH) library, and so on.	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	Must link
sgx_tkey_exchange.lib	Trusted key exchange library	Optional
sgx_tprotected_fs.lib	Protected File System library	Optional
sgx_tedmm.lib	Support for Intel® SGX Enclave Dynamic Memory Management (Intel® SGX EDMM) on Intel® SGX 2.0 hardware platforms	Optional, Import sgx_tedmm.edl
libsgx_tswitchless.asgx_tswitchless.lib	Switchless Enclave Function Calls	Optional
sgx_pcl.lib	Enables Intel® SGX Protected Code Loader for enclave code confidentiality	Optional

## Simulation Libraries

The Intel® SGX 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 EENTER. The trusted simulation library is primarily responsible for simulating the Intel® SGX instructions that can execute inside an enclave: EEXIT, EGETKEY, and EREPORT.

---

### NOTE

Simulation mode does not require the Intel SGX support in the CPU. However, the processor must support the Intel® Streaming SIMD Extensions 4.1 instructions at least.

---

## Linking Application with Untrusted Libraries

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

Table 9 Untrusted Libraries included in the Intel® SGX SDK

Name	Description	Comment
<code>sgx_urts.lib</code>	Provides functionality for applications to manage enclaves	Must link when running in HW mode.  <code>sgx_urts.dll</code> is included in Intel® SGX PSW
<code>sgx_urts_simulation.dll</code>	uRTS library used in simulation mode	Dynamically linked
<code>sgx_urts_simulation.lib</code>	The counterpart of <code>sgx_urts.lib</code> for simulation mode	Must link when running in simulation mode
<code>sgx_uae_service.lib</code>	Provides both enclaves and untrusted applications access to services provided by the AEs	Must link when running in HW mode.  <code>sgx_uae_service.dll</code> is included in Intel® SGX PSW

<code>sgx_uae_service_sim.dll</code>	Untrusted AE support library used in simulation mode	Dynamically linked
<code>sgx_uae_service_sim.lib</code>	The counterpart of <code>sgx_uae_service.lib</code> for simulation mode	Must link when running in simulation mode
<code>sgx_ukey_exchange.lib</code>	Untrusted key exchange library (built with /MD)	Optional
<code>sgx_ukey_exchangemt.lib</code>	Untrusted key exchange library (built with /MT)	Optional
<code>sgx_status.dll</code>	Provides functionality for applications to register Enclave Signing Key White List Certificate Chain	Optional
<code>sgx_capable.dll</code>	Provides functionality for applications to query Intel® SGX device status and the version of PSW installed, or to enable the Intel® SGX device.	Optional
<code>sgx_uprotected_fs.lib</code>	Provides implementation for system calls outside an enclave required by Intel® Protected File System library.	Optional

### Loading Untrusted Intel® SGX DLLs

The Intel® SGX DLLs shipped with the Intel® SGX PSW (`sgx_urts.dll` and `sgx_uae_service.dll`) are installed in the system directory. You must lock down the Intel® 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 Intel® SGX DLL in that directory. If the application implicitly loads the Intel® SGX DLLs, then the bad copy will get loaded before the original Intel® SGX DLLs from the system path.

To make sure that an Intel® SGX application is loading the Intel® 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`

## 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. [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 header files if any
        //Will be includedd in enclave_t.h

        //Trusted function prototypes

    };

    untrusted {
        //Include header files if any
        //Will be included in enclave_u.hhead

        //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 header files if any
    //Will be included in enclave_t.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 header files if any
    //Will be included in enclave_u.h
    //Will be inserted in untrusted header file
    "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 the following example, the enclave will use the `stdio.h` from the Intel® Software Guard Extensions 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 the Intel SGX 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), 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

<b>Data Types</b>					
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			
<b>Pointer Parameter Handling</b>					
in	out	user_check	count	size	readonly
isptr	string	wstring			
<b>Others</b>					
enclave	from	import	trusted	untrusted	include
public	allow	isary	const	propagate_errno	transition_using_threads
<b>Function Calling Convention</b>					
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 **64-bit** `long double`.

Basic data types can be modified using the C modifiers:

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

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

### Pointers

EDL defines several attributes that can be used with pointers:

```
in, out, user_check, string, wstring, size, count,
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 and application code.

---

**Pointer Handling**

Pointers should be decorated with either a pointer direction attribute `in`, `out` or a `user_check` attribute explicitly. 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` or `count` modifier. If neither of these 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 at runtime, respectively, and the ECALL and OCALL functions will not be executed.

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, you have to validate and copy the buffer contents, recursively if needed, yourself. Alternatively, you can define a structure that can be deep copied. See [Structure Deep Copy](#) for more information.

### 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);
    };
};
```

### Unsupported Syntax:

```
enclave {
    trusted {
        // Pointers without a direction attribute
        // or 'user_check' are not allowed
        public void test_ecall_not(int * ptr);

        // 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 the pointer 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 the 'application' side (untrusted side). Content pointed to by `ptr`, one integer value, will be copied into the newly allocated memory outside. Any changes performed by the application will 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 the application 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`, one integer value, 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 behavior of wrapper functions when using the in/out attributes:

**Table 11 Behavior of wrapper functions when using the in/out attributes**

	<b>ECALL</b>	<b>OCALL</b>
user_	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 <code>in</code> and <code>out</code> behavior. Data is copied back and forth.	Same as ECALLs.

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 type with `[isptr]` attribute to indicate that it is a pointer type. See User Defined Data Types for more information.

#### Example:

```
// Error, PVOID is not a pointer in EDL
void foo([in, size=4] PVOID buffer);

// OK
void foo([in, size=4] void* buffer);

// OK, "isptr" indicates "PVOID" is pointer type
void foo([in, isptr, size=4] PVOID buffer);

// OK, opaque type, copy by value
```

```
// Actual address must be in untrusted memory
void foo(HWND hWnd);
```

### Pointer Handling in ECALLs

In ECALLs, the trusted bridge checks that the marshaling structure does not overlap with the 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 the enclave memory. When a pointer to the untrusted memory with the `in` attribute 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 the untrusted memory with the `out` attribute is passed to the enclave, the trusted bridge allocates a buffer in the trusted memory, zeroes the buffer contents to clear any previous data 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 the trusted memory before calling the trusted function, and once the trusted function returns, the trusted bridge copies the contents of the trusted buffer to the untrusted memory. The amount of data copied out is the same as the amount of data copied in.

---

#### **NOTE:**

When an ECALL with a pointer parameter with `out` attribute returns, the trusted bridge always copies data from the buffer in enclave memory to the buffer outside. You must clear all sensitive data from that buffer on failure.

---

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. Attempting to use a buffer allocated by the trusted bridge after it returns results in undefined behavior.

### Pointer Handling in OCALLs

For OCALLs, the trusted proxy allocates memory on the outside stack to pass the marshaling structure and checks that the pointer parameters with their full range are within enclave. When a pointer to trusted memory with the `in` attribute 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 the untrusted memory. When a pointer to the 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 the 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 the untrusted memory before calling the untrusted function, and after the untrusted function returns the trusted proxy copies the contents of the untrusted buffer to the trusted memory. The amount of data copied out is the same as the amount of data copied in.

When the trusted proxy function returns, it frees all the untrusted stack memory allocated at the beginning of the OCALL function for the pointer parameters with a direction attribute. Attempting to use a buffer allocated by the trusted proxy after it returns results in undefined behavior.

#### Attribute: `user_check`

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 an 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, you must understand the risks associated with passing pointers in and out the enclave, in general, and the `user_check` attribute, in particular. You must ensure that all the pointer checking and data copying are done correctly or risk compromising enclave secrets.

#### 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` are specified.
- If `count` is not specified for the pointer parameter, then it is assumed to be equal to 1, i.e., `count=1`. Then total number of bytes equals to `size`.
- 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.

**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);
    };
};
```

**Unsupported Syntax:**

```
enclave{
    trusted {
        // size/count attributes must be used with
        // pointer direction ([in, out])
        void test_attribute_cant([size=len] void* ptr, size_t len);
    };
};
```

**Attribute: count**

The `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.

### Example

```
enclave{
    trusted {
        // 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);
    };
};
```

### Strings

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, the trusted bridge explicitly NULL terminates the string. The size of the buffer allocated in trusted memory accounts for the length determined in the first step as well as the size of the string termination character.

---

### **NOTE**

There are some limitations on the usage of `string` and `wstring` attributes :

- `string` and `wstring` must not be combined with any other modifier such as `size`, or `count`.
  - `string` and `wstring` cannot be used with `out` alone. However, `string` and `wstring` with both `in` and `out` are accepted.
  - `string` can only be used for char pointers; while `wstring` can only be used for `wchar_t` pointers.
- 

### Example

```
enclave {
    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);

};
};

```

### Unsupported 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);
        void test_string_cant_usercheck([user_check, string] char*
ptr);

        // string/wstring attributes cannot be used
        // with [out] attribute
        void test_string_out([out, string] char* str);

        // string/wstring attributes must be used
        // for char/wchar_t pointers
        void test_string_out([in, string] void* str);

    };
};

```

In the first 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.

### const Keyword

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.

### 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:

#### Unsupported 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 is 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_foo_t val);
    public void test_union2(union_foo_t val);
};

};

```

---

**NOTE:**

When referencing a structure, enum or union inside the EDL file, you must follow C style and use the corresponding key word `struct`, `enum` or `union`.

---

**Structure Deep Copy**

Member pointers in a structure can be decorated with a buffer size attribute `size`, or `count` to indicate deep copy structure instead of shallow copy. When the trusted edge-routines copy the buffer pointed by the structure pointer, they also copy the buffer pointed by the structure member pointer instructed by direction attribute of the structure pointer. The member pointer values are also modified accordingly.

The buffer size of the structure must be a multiple of structure size and the buffer is deep copied as an array of structure. Since function call by value make a shallow copy, deep copy structure is not allowed to call by value. Direction attribute of deep copy structure pointer can be `in` and `in, out`. If a

member pointer is not basic type, trusted edge-routines don't deep copy it recursively .

### Example

```
enclave {
    struct struct_foo_t {
        uint32_t count;
        size_t size;
        [count = count, size = size] uint64_t* buf;
    };

    trusted {
        public void test_ecall_deep_copy([in, count = 1] struct
            struct_foo_t * ptr);
    };
};
```

Before calling the ecall , prepare the following data in untrusted domain as parameter:

```
struct struct_foo_t foo = { 4, 8, data};

foo.count = 4;
foo.size = 8;
foo.buf = address of data[] in untrusted domain.
data[] = {0x1112131415161718,
          0x2122232425262728,
          0x3132333435363738,
          0x4142434445464748}
```

After calling the ecall, the data in trusted domain will be:

```
struct struct_foo_t foo = { 4, 8, data2};

foo.count = 4;
foo.size = 8;
foo.buf = address of data2[] in trusted domain.
data2[] = {0x1112131415161718,
          0x2122232425262728,
          0x3132333435363738,
          0x4142434445464748}
```

---

#### **NOTE:**

When deep copying a pointer parameter with `in` attribute in an OCALL, the pointer in the structure, which is the address of a trusted domain, is copied to untrusted domain ephemeraly. You must avoid this scenario if the address is sensitive data .

---

## Arrays

The Enclave Definition Language (EDL) supports multidimensional, fixed-size arrays to be used in data structure definition and parameter declaration.

Arrays also should be decorated with the attribute `[in]`, `[out]` or `[user_check]` explicitly, which are similar to the pointers.

---

### **NOTE**

Limitations on the array usage:

- `size/count` cannot be used for array types
  - `const` cannot be used for array types
  - Zero-length arrays or flexible arrays are not supported by EDL syntax
  - Pointer arrays are not supported by EDL syntax
- 

### 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]);

    };
};
```

### Unsupported 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]);

    };
};
```

A special attribute `isary` is used to designate function parameters that are of a user defined type array. See User Defined Data Types for more information.

## 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`. 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 a 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.
- 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.

---

### NOTE

`isptr`, `isary` and `readonly` can only be used to decorate a user defined data type. Do not use them for any basic types, pointers or arrays.

`readonly` can only be used with `isptr` attribute. Any other usage of `readonly` is not allowed.

---

### Example

```
enclave {
    include "user_types.h" // for typedef void * pBuf;
                          // and typedef void const * pBuf2;
                          // and typedef int uArray[10];

    trusted {

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

        public void test_isptr_readonly(
            [in, isptr, readonly, size=len] pBuf2 pBuf2ptr,
            size_t len);
    }
}
```

```

        public void test_isary([in, isary] uArray arr);
    };
};

```

### Unsupported Syntax:

```

enclave {
    include "user_types.h" //for typedef void const * pBuf2;
                          // and typedef int uArray[10];

    trusted {
        // Cannot use [out] when using [readonly] attribute
        void test_isptr_readonly_cant(
            [in, out, isptr, readonly, size=len] pBuf2
            pBuf2ptr,
            size_t len);

        // isptr cannot be used for pointers/arrays
        public void test_isptr_cant1(
            [in, isptr, size=len] pBuf* pBufptr,
            size_t len);
        public void test_isptr_cant2(
            [in, isptr, size=len] void* pBufptr,
            size_t len);

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

        // size/count attributes cannot be used for user-defined
        array types
        public void test_isary_cant_size(
            [in, size=len] uArray arr,
            size_t len);,

        // isary cannot be used for pointers/arrays
        public void test_isary_cant(
            [in, isary] uArray arr[4]);

    };
};

```

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 `pBuf2ptr` is `readonly`, so you cannot use the `out` attribute.

### Preprocessor Capability

The EDL language supports macro definition and conditional compilation directives. To provide this capability, the `sgx_edger8r` first uses the compiler preprocessor to parse the EDL file. Once all preprocessor tokens have been

translated, the `sgx_edger8r` then parses the resulting file as regular EDL language. This means that developers may define simple macros and use conditional compilation directives to easily remove debug and test capabilities from production enclaves, reducing the attack surface of an enclave. See the following EDL example.

```
#define SGX_DEBUG

enclave {
    trusted {
        // ECALL definitions
    }
    untrusted {
        // OCALL definitions
#ifdef SGX_DEBUG
        void print([in, string] const char * str);
#endif
    }
}
```

The current `sgx_edger8r` does not propagate macro definitions from the EDL file into the generated edge-routines. As a result, you need to duplicate macro definitions in both the EDL file as well as in the compiler arguments or other source files.

We recommend you only use simple macro definitions and conditional compilation directives in your EDL files.

The `sgx_edger8r` starts searching the PATH for the Intel® compiler. If it does not find the Intel compiler, then it searches for the Microsoft\* Visual Studio compiler. The `sgx_edger8r` uses the first compiler it finds. However, if the `sgx_edger8r` does not find any compiler in the PATH, it then looks for the Visual Studio compiler in the registry. If the `sgx_edger8r` still cannot find a compiler, it will then generate a warning message and will parse the EDL directly. to parse macros and conditional compilation directives that might be in the EDL file. You may override the default search behavior or even specify a different preprocessor with the `--preprocessor` option.

### 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
<code>cdecl</code>	Caller	Pushes parameters on the stack (right to left)
<code>stdcall</code>	Callee	Pushes parameters on the stack (right to left)
<code>fastcall</code>	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 {
    include "sgx_stdio_stubs.h" //for FILE and other definitions

    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 * buf-
            fer,
            size_t size,
            size_t count,
            [user_check]FILE * stream);

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

        [stdcall] void test_std_call(void);
    }
}
```

```
};
};
```

### Unsupported 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);
    };
};
```

### Propagating errno in OCALLs

OCALLs may use the `propagate_errno` attribute. When you use this attribute, the `sgx_edger8r` produces slightly different edge-routines. The `errno` variable inside the enclave, which is provided by the trusted Standard C library, is overwritten with the value of `errno` in the untrusted domain before the OCALL returns. The trusted `errno` is updated upon OCALL completion regardless whether the OCALL was successful or not. This does not change the fundamental behavior of `errno`. A function that fails must set `errno` to indicate what went wrong. A function that succeeds, in this case the OCALL, is allowed to change the value of `errno`.

### Example

```
enclave {

    include "sgx_stdio_stubs.h" //for FILE and other definitions

    trusted {
        public void test_file_io(void);
    };

    untrusted {

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

```

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

[cdecl, dllimport] size_t fwrite(
    [in, size=size, count=count] const void * buf-
    fer,
    size_t size,
    size_t count,
    [user_check]FILE * stream) propagate_errno;
};
};

```

### Importing EDL Libraries

You can implement export and import functions in external trusted libraries, akin to static libraries in the untrusted domain. To add these functions to an enclave, use the enclave definition language (EDL) library import mechanism.

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

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. It is recommended to use different names to distinguish the library EDL file and the enclave 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);
    };
};
```

### **Granting Access to ECALLs**

The default behavior is that ECALL functions cannot be called by any of the untrusted functions.

To enable an ECALL to be directly called by application code as a root ECALL, the ECALL should be explicitly decorated with the `public` keyword to be a

public ECALL. Without this keyword, the ECALLs will be treated as private ECALLs, and cannot be directly called as root ECALLs.

### Syntax

```
trusted {
    public <function prototype>;
};
```

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.

To grant an OCALL function access to an ECALL function, specify this access using the `allow` keyword. Both public and private ECALLs can be put into the allow list.

### Syntax

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

### Example

```
enclave {
    trusted {
        public void clear_secret();
        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, clear_secret);
    };
};
```

In the above example, the untrusted code is granted different access permission to the ECALLs.

ECALL	called as root ECALL	called from <code>replace_secret</code>
<code>clear_secret</code>	Y	Y
<code>get_secret</code>	Y	N
<code>set_secret</code>	N	Y

## Using Switchless Calls

ECALLs and OCALLs can use the **transition\_using\_threads** attribute as a postfix of the function declaration in the EDL file. When you use this attribute, the `sgx_edger8r` produces different edge-routines.

ECALLs and OCALLs with the **transition\_using\_threads** attribute use the Switchless mode of operation to serve the call.

(See: [Using Switchless Calls](#))

### Example

```
enclave {
    trusted {
        public void ecall_empty(void);
        public void ecall_empty_switchless(void) transition_using_
            threads;
    };
    untrusted {
        void ocall_empty(void);
        void ocall_empty_switchless(void) transition_using_threads;
    };
};
```

## Enclave Configuration File

The enclave configuration file is an XML file containing the user defined parameters of an enclave. This XML file is a part of the enclave project. 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>
    <EnableKSS>1</EnableKSS>
    <ISVEXTPRODID_H>1</ISVEXTPRODID_H>
    <ISVEXTPRODID_L>2</ISVEXTPRODID_L>
    <ISVFAMILYID_H>3</ISVFAMILYID_H>
    <ISVFAMILYID_L>4</ISVFAMILYID_L>
</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 is used.

**Table 13 Enclave Configuration Default Values**

<b>Tag</b>	<b>Description</b>	<b>Default Value</b>
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.	0x1000000
DisableDebug	Enclave cannot be debugged.	0 - Enclave can be debugged
MiscSelect	The desired Extended SSA frame feature.	0
MiscMask	The mask bits of MiscSelect to enforce.	0xFFFFFFFF
EnableKSS	Enable the Key Separation and Sharing feature	0
ISVEXTPRODID_H	ISV assigned Extended Product ID (High 8 bytes)	0
ISVEXTPRODID_L	ISV assigned Extended Product ID (Low 8 bytes)	0
ISVFAMILYID_H	ISV assigned Family ID (High 8 bytes)	0
ISVFAMILYID_L	ISV assigned Family ID (Low	0

	8 bytes)	
--	----------	--

`MiscSelect` and `MiscMask` are for future functional extension. Currently, `MiscSelect` must be 0. Otherwise the corresponding enclave may not be loaded successfully.

Set `EnableKSS` to 1 to enable the Key Separation & Sharing (KSS) feature for the enclave. `ISVEXTPRODID_H` and `ISVEXTPRODID_L` are used to set the ISV assigned Extended Product ID, which is a 16-byte value. `ISVFAMILYID_H` and `ISVFAMILYID_L` are for the 16-bytes ISV assigned Family ID. Note that you need to enable KSS before setting the ISV assigned Extended Product ID and the ISV assigned Family ID.

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 **Intel® SGX Configuration** is provided for users to edit their configuration file conveniently. See [Using Intel® SGX Configuration Add-in](#) for details.

If there is no enough stack for the enclave, ECALL returns the error code `SGX_ERROR_STACK_OVERRUN`. This error code gives the information to enclave writer that the `StackMaxSize` may need further adjustment.

<https://www.intel.com/content/www/us/en/content-details/736463/white-paper-asynchronous-enclave-exit-notify-and-the-edeccssa-user-leaf-function.html>.

## Enclave Project Configurations

Depending on the development stage you are at, 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 cannot be added to the allowlist.

- **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 cannot be added to the allowlist 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 key that has been added to the allowlist.

For additional information on the different enclave signing methods, see [Enclave Signing Tool](#) and [Enclave Signer Usage Examples](#)

### Loading and Unloading an Enclave

Enclave source code is built as a dynamic link library. To use an enclave, the `enclave.dll` should be loaded into protected memory by calling the API `sgx_create_enclave()` or `sgx_create_enclave_ex()`. The `enclave.dll` must be signed by `sgx_sign.exe`. When loading an enclave for the first time, the loader gets a launch token and saves it back to the `in/out` parameter `token`. You 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

file. Providing a valid launch token can enhance the load performance. To unload an enclave, 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,
        &updated, &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 Intel 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 Intel 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 Intel® 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`.

---

### Using Switchless Calls

An enclave switch occurs whenever the execution of a CPU jumps in (EENTER) or out (EEXIT) of an enclave; for example, when making ECALLs/OCALLs. Enclave switches have a performance overhead. For workloads with short and frequent calls, the enclave switching overhead can be reduced using Switchless Calls. Switchless Calls introduce a new mode of operation to perform calls from/to Intel® SGX enclaves, using worker threads inside and outside the enclave.

---

**PERFORMANCE NOTE:**

Switchless calls is an advanced feature. It requires additional worker threads and configuration, performance testing and tuning. It should be used for

---

---

workloads that require fine performance tuning. Misconfiguration may result in under utilized worker threads, which consumes CPU time while not serving any tasks.

---

### Usage

To use Switchless calls, the EDL attribute `transition_using_threads` should be postfixed to the ECALLs and OCALLs where Switchless Calls are required. An EDL file can contain ECALLs/OCALLs with or without this attribute.

The application code must create an enclave using `sgx_create_enclave_ex`, set the Switchless flag in an extended options vector, and provide a switchless configuration structure .

On enclave creation, the uRTS creates several trusted and untrusted worker threads according to the Switchless configuration provided via initialization structures and allocates the required data structures for Switchless Calls. Trusted worker threads use regular enclave TCSes. The TCSNum defined in the enclave XML configuration should be updated accordingly when building an enclave with switchless trusted worker threads.

---

### **NOTE:**

You should not use Switchless Calls with TCS binding policy, namely TCSPolicy 0. Using this policy disables concurrent execution of E/OCALLS.

When a developer builds an enclave with the TCS binding policy, they expect the TLS data of the trusted thread to be preserved across calls to the same trusted function. However, this behavior cannot be provided if the enclave uses switchless calls for two main reasons:

- Worker threads handle different switchless ECALLs, despite the TCS binding policy. As a result, the TLS area assigned to any worker thread will be re-used by all the ECALL functions that the worker thread services.
- When a switchless call request times out, it is serviced as a regular ECALL using a TCS reserved for regular ECALLs. Thus, the switchless ECALL will re-use the TLS area of a regular ECALL.

---

Example usage of `sgx_create_enclave_ex`:

```
sgx_launch_token_t token = {0};
sgx_status_t ret = SGX_ERROR_UNEXPECTED;
```

```
int updated = 0;

sgx_uswitchless_config_t us_config = { 0, 1, 1,
100000, 100000, { 0 } };

void* enclave_ex_p[32] = { 0 };

enclave_ex_p[SGX_CREATE_ENCLAVE_EX_SWITCHLESS_
BIT_IDX] = &us_config;

sgx_enclave_id_t eid;

const char* fname = "enclave.signed.so";

ret = sgx_create_enclave_ex ( fname,

    SGX_DEBUG_FLAG, &token, &updated, &eid,
    NULL,

    SGX_CREATE_ENCLAVE_EX_SWITCHLESS,
    enclave_ex_p );
```

## High Level Overview

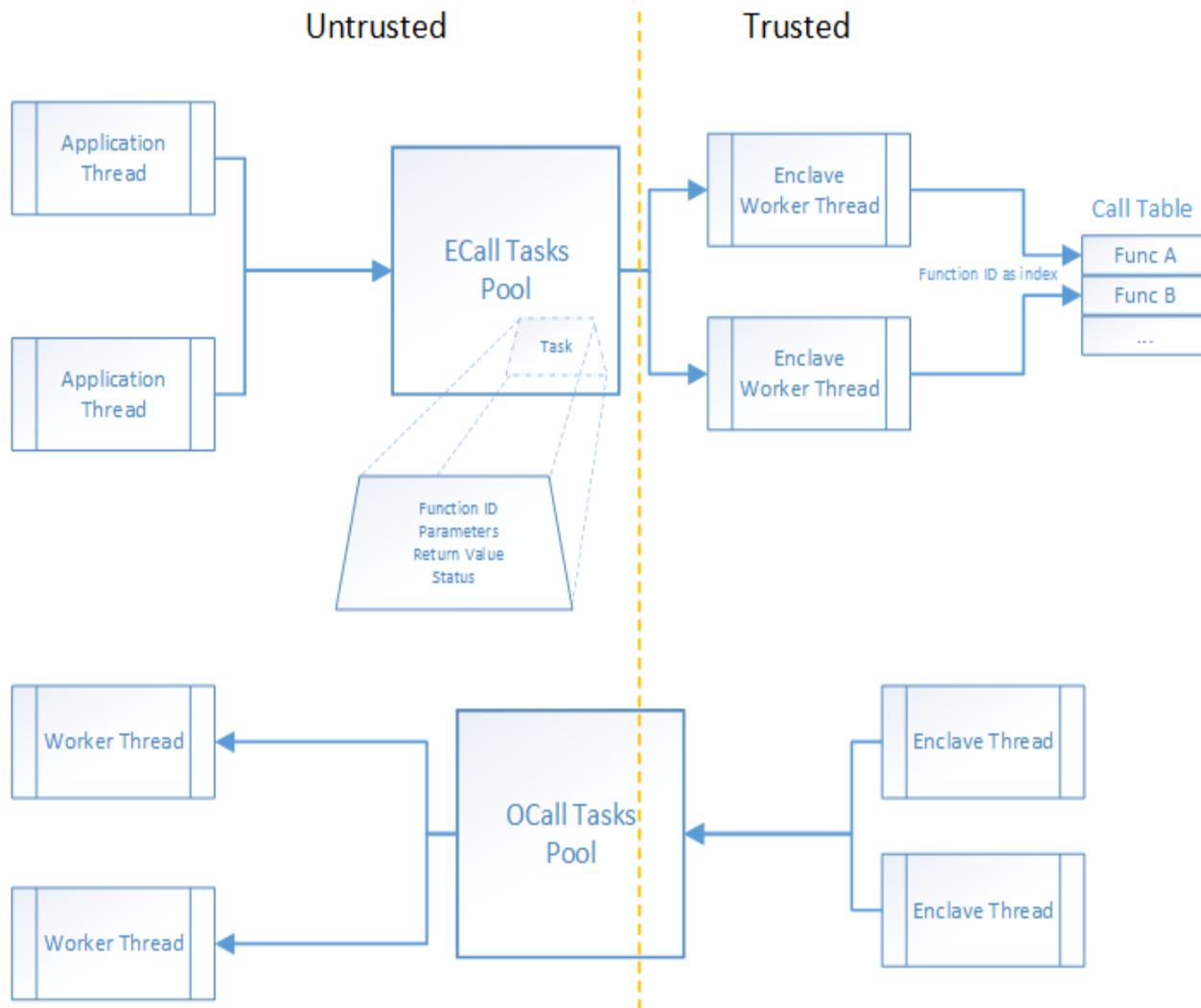


Figure 12 Switchless Calls Architecture

**Major Highlights:**

- Two task pools, for ECALL and OCALL-tasks respectively.
- Several worker threads servicing ECALLs (running inside the enclave) and OCALLs (running outside the enclave) requests.
- Task objects describing an ECALL/OCALL request including all the required parameters.

**Task Pool**

The Task Pool contains task requests pending to be executed. It uses task objects to transfer data between trusted and untrusted sides without invoking

EENTER/EEXIT.

The application defines at runtime the task pool size, namely the number of concurrent task requests.

### Worker Threads

Worker threads wait for one or more pending tasks in the relevant task pool and start executing the tasks until the task pool gets empty.

The number of working threads is defined by the application at runtime and should be at least two to support nested Switchless Calls.

The depth of nested Switchless Calls (Switchless OCALL → Switchless ECALL → ... → Switchless OCALL) cannot be greater than the number of working threads.

### Fallback to regular ECALLs/OCALLs

When the Task Pool is full, or when all Worker Threads are busy, a Switchless Call falls back to a regular ECALL/OCALL.

### Nested Switchless ECALL

Switchless Calls do not support private nested ECALLs. Nested ECALLs using the `transition_using_threads` keywords must be public as well. Allowing a nested switchless ECALL is not sufficient. A non-public nested ECALL returns `SGX_ERROR_ECALL_NOT_ALLOWED` to the application.

### Switchless Calls Usage Configuration Tips

The Switchless Calls operation mode can improve performance of some workloads. However, this mode is complex and may cause a slowdown or/and a resource overloading by busy-wait worker threads.

It is highly recommended to introduce performance measurements and tuning to the development cycle when you use switchless calls. A simple example is shown in the Switchless sample code.

### Switchless Calls Operation Mode Callbacks

The application can register callbacks for worker thread events. Worker threads can send four types of events:

- worker threads starts
- worker thread exits
- worker thread enters idle state (sleep)
- worker thread misses switchless call (fallback)

Worker thread events contain statistics of processed and missed (fallback) switchless calls. The statistics is common for all worker threads of the same type (trusted/untrusted).

Application may use the Switchless mode callbacks (`sgx_uswitchless_worker_callback_t`) to gather additional performance data. Note that the worker thread MISS event (`SGX_USWITCHLESS_WORKER_EVENT_MISS`) may happen and cause additional overhead.

Applications that use Switchless Calls may find it useful to detect the HW capabilities of the CPU: the number of cores and threads to configure the switchless configuration structure.

The worker thread START event (`SGX_USWITCHLESS_WORKER_EVENT_START`) can be used to set thread affinity .

See the example of the worker thread exit callback below. For the callback prototype, refer to [sgx\\_uswitchless\\_worker\\_callback\\_t](#).

```
// global processed/missed calls counters
// 0,1 - untrusted; 2,3 - trusted
uint32_t g_stats[4] = { 0 };

/**
 * callback to log switchless calls stats
 */
void exit_callback (
    sgx_uswitchless_worker_type_t type,
    sgx_uswitchless_worker_event_t event,
    const sgx_uswitchless_worker_stats_t* stats )
{
    // last thread exiting will update the latest results
    g_stats[type*2] = stats->processed;
    g_stats[type*2+1] = stats->missed;
}
```

### Enabling Enclave Code Confidentiality

Intel® Software Guard Extensions Protected Code Loader (Intel® SGX PCL) is intended to protect Intellectual Property (IP) within the code for Intel® SGX enclave applications running on the Windows\* OS.

**Problem:** Intel® SGX provides integrity of code and confidentiality and integrity of data at run-time. However, it does NOT provide confidentiality of code offline as a binary file on disk. Adversaries can reverse engineer the binary enclave DLL.

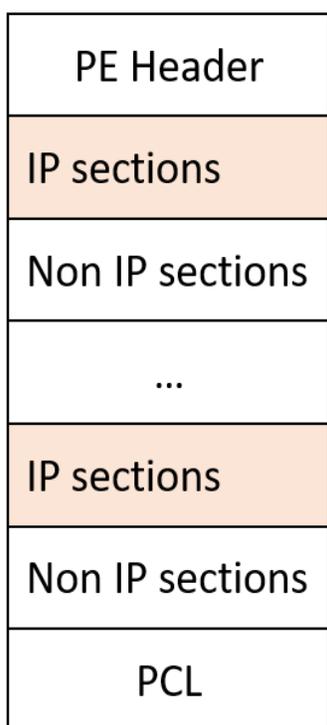
**Solution:** Encrypt the enclave DLL at build time and decrypt it at enclave load time.

### Intel® SGX PCL Architectural Overview

#### Build Time:

Figure below shows the Intel® SGX PCL build flow.

#### Original ISV Enclave



#### Encrypted ISV Enclave

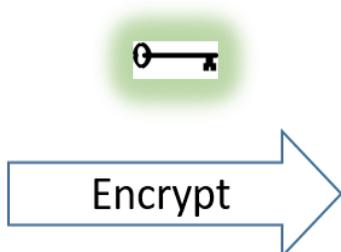
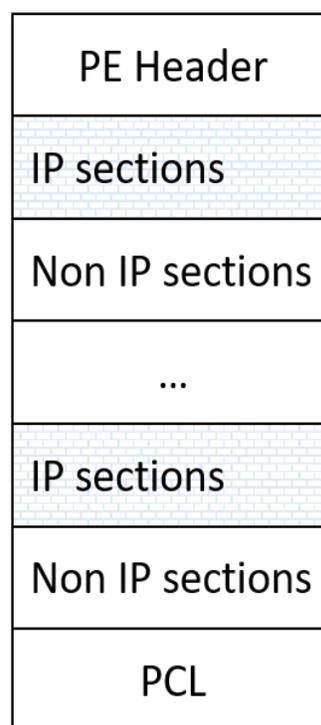


Figure 13 Intel® SGX PCL Build Flow

1. The Intel® SGX PCL library is linked into the ISV Intel® SGX IP Enclave.

2. Before the ISV IP Enclave is signed, the linked DLL is modified so that PE sections containing IP are encrypted. The green key designates the symmetric encryption/decryption key.

#### Notes:

- The Intel® SGX PCL encryption tool treats all sections as IP, except for sections that are required by either the signing tool, the Intel® SGX PSW Enclave Loader, or the Intel® SGX PCL decryption flow. For a detailed list, see 'Regions that are Not Encrypted' below.
- Encryption/decryption key management is the ISV responsibility, which is out of scope for this document.

#### Run Time

##### ISV Sealing Enclave

To load an IP Enclave, the ISV must first transport a decryption AES key to the user local machine, seal it on the user local machine, and use it as an input for the Intel® SGX PCL. For this, the ISV must devise the second enclave, the 'Sealing Enclave'. The figure below shows this flow:

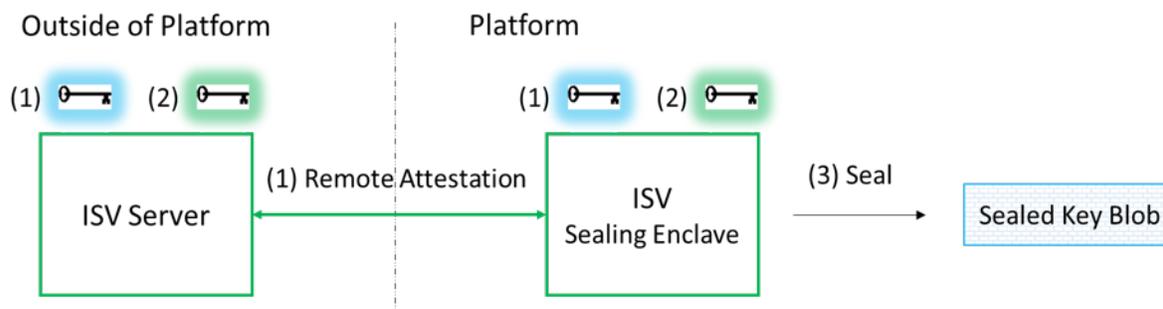


Figure 14 ISV Sealing Enclave Flow

The ISV Sealing Enclave performs the following operation:

1. Uses the existing standard Intel® SGX SDK Remote Attestation to generate a secure session with the ISV server. (The light blue key illustrates session keys)

2. Receives the decryption key from the ISV server in a secured way.

See details at "Intel® SGX PCL Decryption key provisioning" below. (The green key illustrates a decryption key)

3. Uses the existing standard Intel® SGX SDK sealing mechanism to generate the sealed key and store it locally.

**Notes:**

- For the Sealing Enclave and the IP Enclave to be able to seal and unseal the decryption key, both enclaves must be signed with the same Intel® SGX ISV signing key and have the same ProdID.
- Once the sealed key is generated, it can be stored in nonvolatile memory on the platform. This decreases the number of remote attestations required to run.

**ISV IP Enclave**

Figure below shows the enclave loading flow:

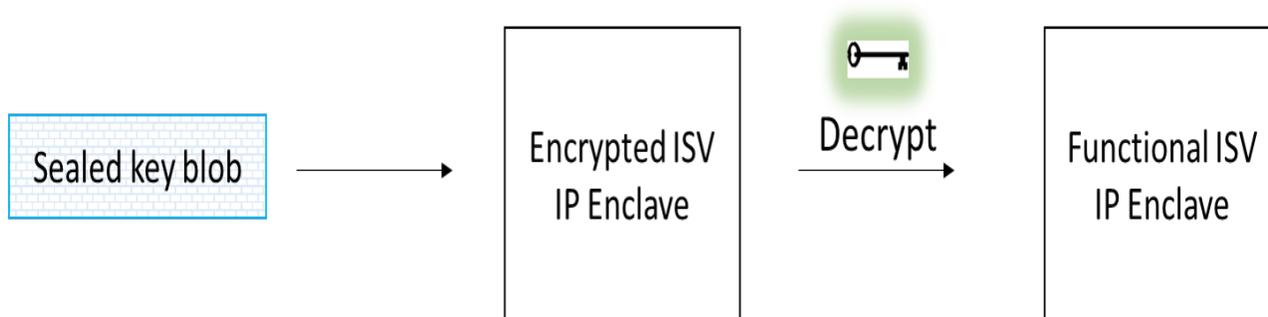


Figure 15 ISV IP Enclave Loading Flow

The ISV IP Enclave performs the following operations:

1. Receives the Sealed Key Blob as input.
2. Unseals the blob to receive the decryption key.
3. Uses the decryption key to decrypt the IP content.

**Comparison with Standard Flow**

Table below summarizes the differences between the IP Enclave load flows [with](#) and without the Intel® SGX PCL.

Table 14 Comparison of flows with and without Intel® SGX PCL:

Step	Standard Flow (No Intel® SGX PCL)	Intel® SGX PCL Flow
Build Time	<ul style="list-style-type: none"> <li>• <b>Link:</b> ISV libraries and objs are linked to Enclave.dll</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Link:</b> ISV libraries, objs, and <a href="#">sgx_pcl.lib</a> are linked to IPEnclave.dll</li> <li>• <b>Encrypt</b> IPEnclave.dll to IPEn-</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>Sign:</b> Enclave.dll is signed to generate Enclave.signed.dll</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Sign:</b> IPEnclave.enc.dll is signed to generate IPEnclave.signed.dll</li> </ul>
Enclave Load	<ol style="list-style-type: none"> <li>1. Enclave application loads the enclave using <code>sgx_create_enclave</code>.</li> <li>2. <code>sgx_create_enclave</code> performs an implicit ecall.</li> <li>3. The implicit ecall initiates an enclave runtime initialization flow.</li> </ol>	<ol style="list-style-type: none"> <li>1. <b>Enclave application gets the sealed decryption key.</b></li> <li>2. Enclave application loads the enclave using <code>sgx_create_enclave_ex</code>, providing the sealed decryption key</li> <li>3. <code>sgx_create_enclave_ex</code> performs an implicit ecall.</li> <li>4. <b>The Implicit ecall invokes the Intel® SGX PCL flow.</b></li> <li>5. <b>Intel® SGX PCL unseals the sealed blob to get the decryption key.</b></li> <li>6. <b>Intel® SGX PCL decrypts the encrypted IP sections and returns the enclave to its functional state.</b></li> <li>7. The process continues with the enclave runtime initialization flow.</li> </ol>

**Note:** Intel® SGX PCL does not support the Simulation mode.

## Security Considerations

### Not Encrypted Regions

The ISV must make sure the regions described in the table below do not contain the ISV IP. The encryption tool will **NOT** encrypt these regions.

Table 15 Not Encrypted Regions

Region	Description
Content outside the PE section	Encryption tool does not encrypt content that is outside a section
Section ".reloc"	Intel® SGX PCL uses ".reloc" section to apply relocations to its code and data
Sections "niptx", "nipdt", "nipdt1", "nipdt2" and "pcltbl"	These sections include code and data used during the Intel® SGX PCL flow
Sections ".debug", "sgxmeta", "sgxdata", "sgxvers", ".tls" and ".edata"	These sections are required by either the Intel® SGX SDK signing tool or the Intel® SGX PSW

	enclave loader
Export directory, export directory strings of symbols, export directory functions, export directory names, export directory ordinals, TLS directory, thread data, debug directories, debug data	These regions are required by either the Intel® SGX SDK signing tool or the Intel® SGX PSW enclave loader

### Writable Sections

The Intel® SGX PCL encryption tool sets the Writable bit in the section flags of the encrypted PE sections. As a result, all pages that belong to such PE sections, including portions of the enclave code and read-only data, are writable at enclave runtime.

### Intel® SGX PCL Cryptographic Standards

At build time, the encryption tool uses:

- SHA256 to compute the hash of the symmetric encryption/decryption key and embeds it into the IP enclave binary.
- AES-GCM-128 to encrypt-in-place the IP sections.
- RDRAND to generate the per-section random IVs.

At run time the Intel® SGX PCL uses:

- SHA256 to compute the hash of the unsealed symmetric encryption/decryption key. The Intel® SGX PCL verifies the integrity of the symmetric encryption/decryption key by comparing its hash with the one embedded in the IP enclave binary at build time.
- AES-GCM-128 to decrypt-in-place the IP sections.

### Intel® SGX PCL Crypto Code Snippets from OpenSSL

Intel® SGX PCL library includes code snippets from OpenSSL1.1.1 (with slight modifications to enable running with Intel® SGX PCL). Those snippets are now part of the ISV's IP enclave's TCB. If in the future, an identified vulnerability in OpenSSL1.1.1 requires modification to a file from which these snippets originate, ISV must update the snippets accordingly.

### Integrating Intel® SGX PCL with an existing Intel® SGX solution

Integrating an ISV enclave with the Intel® SGX PCL requires the ISV to apply modifications to the ISV solution:

1. Apply modifications to the IP enclave.
2. Apply modifications to the enclave application that loads the enclave(s).
3. Create an additional enclave, the Sealing Enclave.

**Note:** Steps above are already applied to SampleEnclavePCL. See README.md for instructions on building and running the sample code.

**Disclaimer:** This chapter presents a pseudo code, which is not secure, not complete, and it will not compile. For the complete code, see SampleEnclavePCL.

### Modifications to IP Enclave

- Disable incremental linking if enabled.
- Disable whole program optimization if enabled.
- Disable interprocedural optimization "Multi File" if enabled.
- Disable Control Flow Guard if enabled
- Add `sgx_pcl.lib` to the list of static libraries that the enclave is linked to.
- Change the build flow so that the build time encryption tool `sgx_enc_ip.exe` is run before the enclave is signed  

```
sgx_enc_ip.exe -k key.bin -i IPEnclave.dll -o IPEnclave.enc.dll
```
- Change the signing command to sign the encrypted enclave `IPEnclave.enc.dll`
- No modifications are required for the IP Enclave source code.

### Modifications to Enclave Application

Required steps:

1. Get the sealed blob:
  - If a file containing the sealed blob exists (for example, it was generated during the previous runs), read it.
  - If the file does not exist:
    - Create the Sealing Enclave.
    - Use the Sealing Enclave to provision the decryption key to the platform and seal it.
    - Save the sealed key to a file on the platform for future use.

2. Load the encrypted enclave using `sgx_create_enclave_ex` and provide it with the sealed blob.

Pseudo code:

```
#define SEALED_KEY_FILE_NAME "SealedKey.bin"

#define IP_ENCLAVE_FILE_NAME "IPEnclave.signed.dll"

#define SEALING_ENCLAVE_FILE_NAME "SealingEnclave.signed.dll"

uint8_t* sealed_key;
size_t sealed_key_size;

if(file_exists(SEALED_KEY_FILE_NAME))
{
    // Sealed key file exists, read it into buffer:
    ReadFromFile(SEALED_KEY_FILE_NAME, sealed_key);
}
else
{
    /*
    * Sealed key file does not exist. Create it:
    * 1. Create the Sealing Enclave
    * 2. Use the Sealing Enclave to provision the decryption key
    * onto the platform and seal it.
    * 3. Save the sealed key to a file for future uses
    */
    // 1. create the sealing enclave
    sgx_create_enclave(
        SEALING_ENCLAVE_FILE_NAME,
        debug,
        &token,
        &updated,
        &seal_enclave_id,
        NULL);

    /*
    * 2. Use the Sealing Enclave to provision the decryption key
    * onto the platform and seal it:
    */
}
```

```

*/
ecall_get_sealed_key_size(seal_enclave_id, &sealed_key_size);
sealed_key = (uint8_t*)malloc(sealed_key_size);
ecall_get_sealed_key(seal_enclave_id, sealed_key, sealed_key_size);
// 3. Save the sealed key to a file for future uses
WriteToFile(SEALED_KEY_FILE_NAME, sealed_key);
}
// Load the encrypted enclave, providing the sealed key:
const void* ex_features[32] = {};
ex_features[SGX_CREATE_ENCLAVE_EX_PCL_BIT_IDX] = sealed_key;
sgx_create_enclave_ex(
    IP_ENCLAVE_FILE_NAME,
    debug,
    &token,
    &updated,
    ip_enclave_id,
    NULL,
    ex_features,
    SGX_CREATE_ENCLAVE_EX_PCL);

```

## Sealing Enclave

### Intel® SGX PCL Decryption key provisioning

This section describes methods for creating and using the ISV Sealing Enclave. The ISV Sealing enclave provisions the decryption key to the user local machine and seals it.

To securely transport the decryption AES key to the user local machine, the ISV Sealing enclave needs to attest to the ISV server, generate a secure session, and use it to provision the decryption key.

### Sending the Intel® SGX PCL Decryption Key from ISV Server to Local Platform

The Remote Attestation sample in this document illustrates and describes in details how to initiate a remote attestation session with an ISV server.

Remote attestation enables the server and the client to share secret keys. Such keys can be used to generate a secure session (for example, using TLS) between the ISV server and the Sealing enclave. The secure session can then

be used to securely provision the decryption key.

### **Sealing the Intel® SGX PCL Decryption Key**

The sealing sample code in this document illustrates how to seal a secret. By default, the Intel® SGX SDK seals the secret using MRSIGNER.

### **Interaction with the Enclave Application**

In the pseudo code above, the ISV Sealing Enclave provides the Enclave Application with the sealed decryption key by implementing the enclave calls `ecall_get_sealed_key_size` and `ecall_get_sealed_key`. This is not an architectural requirement and ISVs can use their own design.

## Intel® Software Guard Extensions SDK Sample Code

After installing the Intel® Software Guard Extensions SDK, you can find the sample code at `[Intel SGX SDK Install Path]src`.

You can open the sample projects in Microsoft\* Visual Studio\* 2019.

- The *SampleEnclave* project shows how to create an enclave.
- The *PowerTransition* project shows how to handle the power transition for an Intel SGX project.
- The *Cxx11SGXDemo* project shows how to use C++11 library inside the enclave.
- 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 *Sgx2Enclave* project shows how to create an Intel® SGX2 enclave and use the `sgx_tedmm` library.

### Sample Enclave

The project *SampleEnclave* shows 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 or OCALLs
- Call trusted libraries inside the enclave

The code is shipped with the Intel® Software Guard Extensions SDK in `[Intel SGX SDK Install Path]src\SampleEnclave`. You can open the project in Microsoft\* Visual Studio 2019.

---

#### **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. See `sgx_enable_device` for additional details.

### Initialize an Enclave

Before establishing any trusted transaction between an application and an enclave, the enclave itself needs to be correctly created and initialized by calling `sgx_create_enclave` provided by the uRTS library.

### Saving and Retrieving the Launch Token

A launch token needs to be passed to `sgx_create_enclave` for enclave initialization. If the launch token was saved in a previous transaction, it can be retrieved and used directly. Otherwise, you can provide an all-0 buffer. `sgx_create_enclave` will attempt to create a valid launch token if the input is not valid. 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.

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

### ECALL/OCALL Functions

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.

### Destroy an Enclave

To release the enclave memory, you need to invoke [sgx\\_destroy\\_enclave](#) provided by the `sgx_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 SDK. You can find the source code in the `[Intel SGX SDK Install Path]src\PowerTransition` directory. The sample code can be built with the Microsoft\* Visual Studio 2019 using the corresponding project in the Microsoft\* Visual Studio 2019.

---

#### **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.

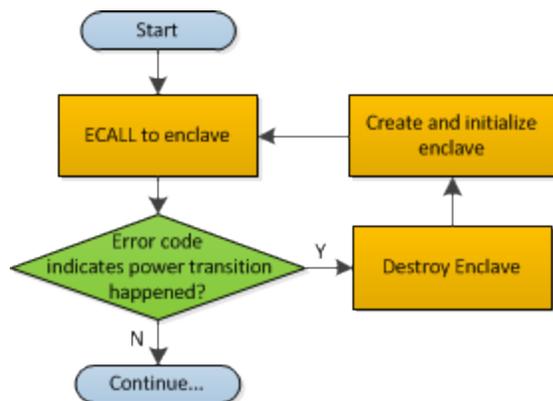


Figure 16 Power Transition Handling Flow Chart

### 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:

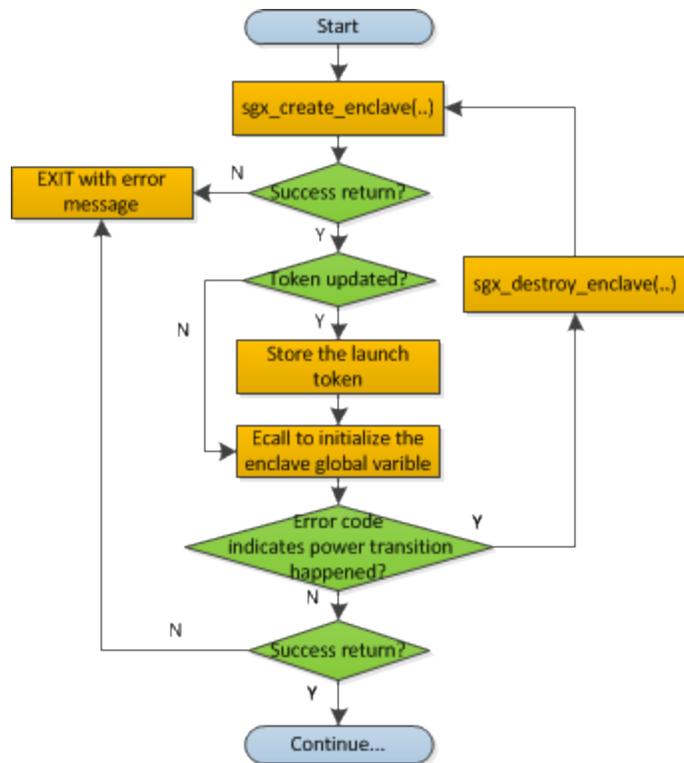


Figure 17 Enclave Initialization ECall after Enclave Creation Flow Chart

`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.

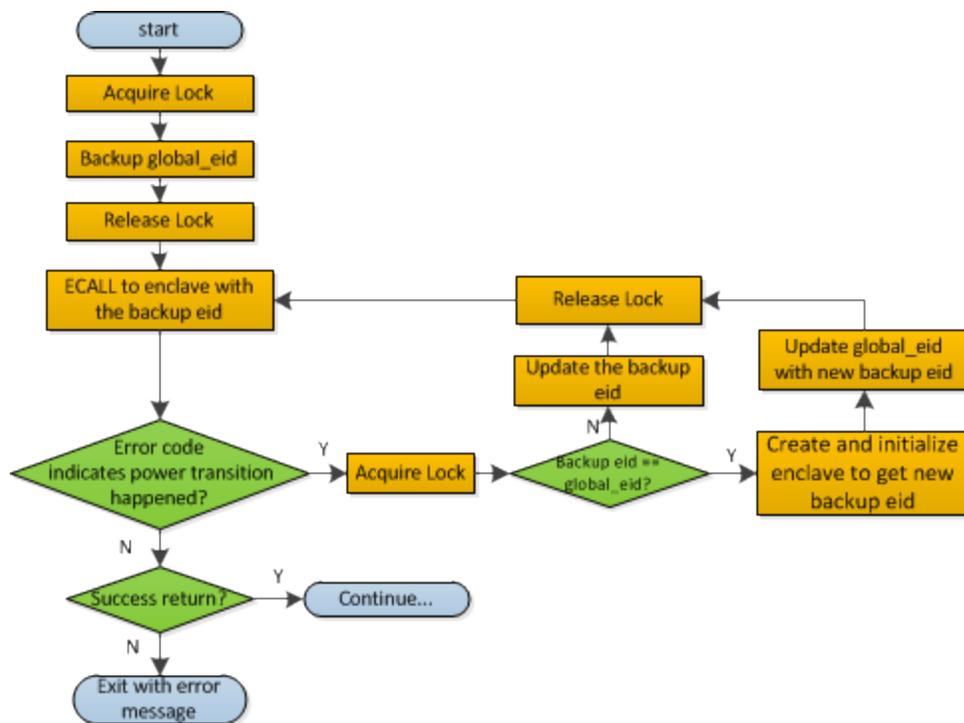


Figure 18 Regular ECALL Flow Chart

**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.

**C++11 Demo**

The project *Cxx11SGXDemo* is designed to illustrate some of the C++11 library features supported inside the enclave provided by the Intel® SGX SDK and the compiler features supported by the Visual Studio 2019 Compiler. This sample provides practical use cases for each of the C++11 features currently supported.

The code is shipped with the Intel SGX SDK and is located in `[Intel SGX SDK Install Path]src\Cxx11SGXDemo`. Open the project in Microsoft® Visual Studio 2019.

**NOTE:**

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

The sample covers a subset of C++11 features inside the enclave as listed in the table below.

**Table 16 Overview of C++11 features covered**

Headers	<pre>#include &lt;typeinfo&gt; #include &lt;functional&gt; #include &lt;algorithm&gt; #include &lt;unordered_set&gt; #include &lt;unordered_map&gt; #include &lt;initializer_list&gt; #include &lt;tuple&gt; #include &lt;memory&gt; #include &lt;atomic&gt; #include &lt;mutex&gt; #include &lt;condition_variable&gt; #include &lt;map&gt;</pre>
Classes	<pre>std::function, std::all_of, std::any_of, std::none_of, std::initializer_list, std::unordered_set, std::unordered_map, std::unordered_multiset, std::unordered_multimap, std::tuple, std::shared_ptr, std::unique_ptr, std::auto_ptr, std::mutex, std::condition_variable</pre>
Compiler features	<pre>lambda expressions, auto, decltype, strongly typed enum classes, range-based for statements, static_assert, new virtual function controls, delegating constructors, variadic templates,</pre>

	substitution failure is not an error (SFINAE), rvalue references and move semantics, nullptr type
--	--

## 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 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 Intel 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 SDK at `[Intel SGX SDK Install Path]src\LocalAttestation` directory. To compile, you only need to open the project with Microsoft\* Visual Studio 2019.

---

### **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.

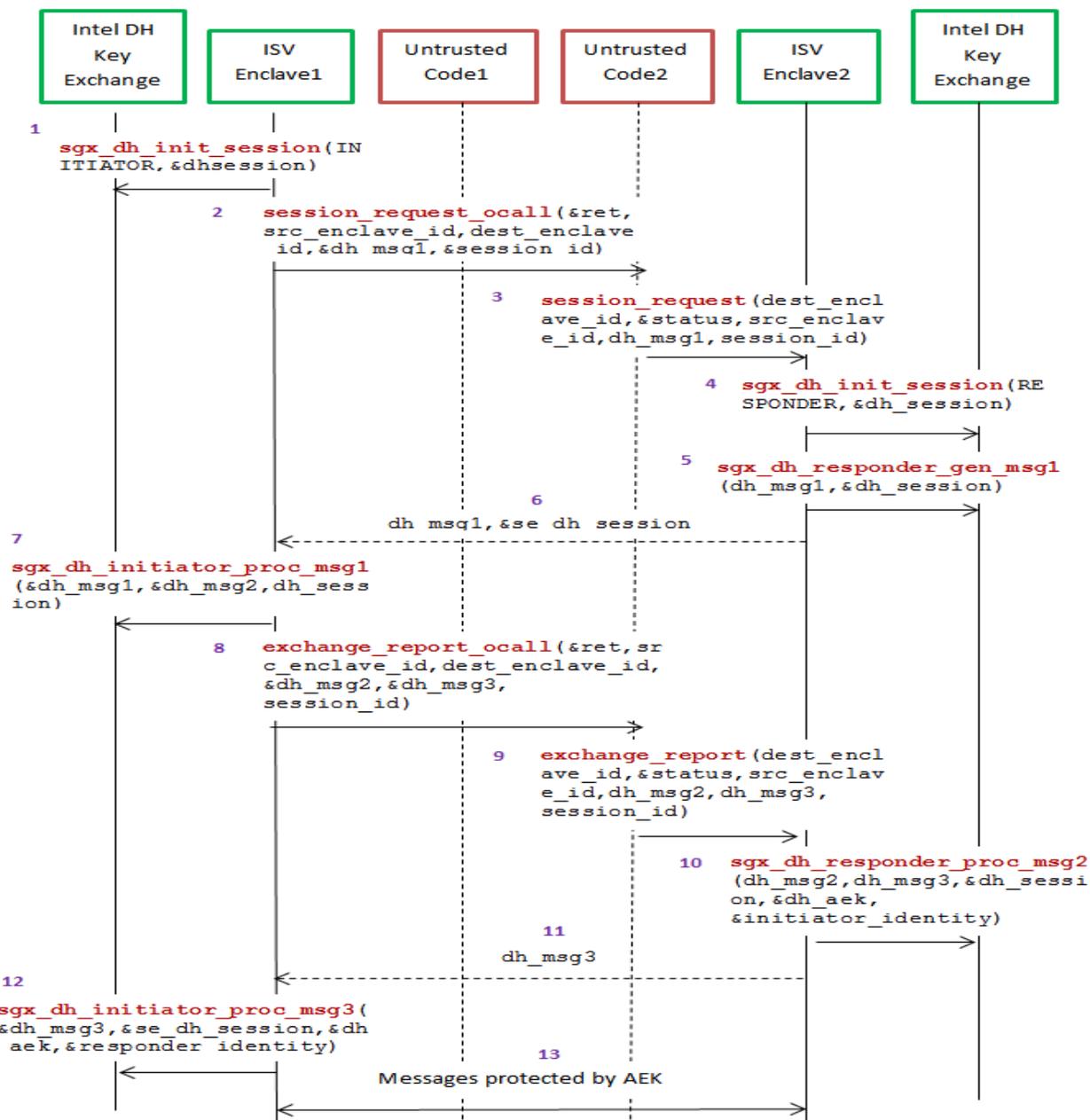


Figure 19 Local Attestation Flow with the DH Key Exchange Library

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

In the figure, ISV Enclave 1 is the initiator enclave and ISV Enclave 2 is the responder enclave.

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

#### Diffie-Hellman Key Exchange Library and Local Attestation 2.0

the Diffie-Hellman (DH) key exchange library also exposes DH key exchange 2.0 APIs for the enclave code to generate and process local key exchange protocol messages. To use DH key exchange 2.0 APIs which are also described in `sgx_dh.h`, add `SGX_USE_LAv2_INITIATOR` to Preprocessor Definitions option.

A local attestation 2.0 flow consists of the steps in previous section except 7 and 10:

1. ISV initiator enclave calls the Intel ECDH key exchange library to initiate the session with the initiator role.
2. The initiator enclave does an OCALL into the untrusted code requesting the Diffie-Hellman Message 1 and session id.
3. The untrusted code does an ECALL into the responder enclave .
4. The responder enclave in turn calls the ECDH key exchange library to initiate the session with the responder role.
5. The responder enclave calls the key exchange library to generate DH Message 1 `ga || TARGETINFO`.
6. DH Message 1 is sent back from the responder enclave to the initiator enclave through an ECALL return to the untrusted code followed by an OCALL return into the initiator enclave.
7. The initiator enclave processes the Message 1 using the key exchange library 2.0 API and generates the DH Message 2 `gb || [Report Enclave 1(h(proto_spec || gb))]SMK` and `report_data` replaced with `proto_spec` which `proto_spec` is `'SGX LA' || Ver || Rev || TARGET_SPEC || padding`.
8. DH Message 2 is sent to the untrusted side through an OCALL.
9. The untrusted code does an ECALL into the responder enclave giving it the DH Message 2 and requesting the DH Message 3.
10. The responder enclave calls the key exchange library 2.0 API to process the DH Message 2 and generates the DH Message 3 `[Report Enclave2(h(ga || proto_spec)) || Optional Payload || ga]SMK`.
11. DH Message 3 is sent back from the responder enclave to initiator enclave through an ECALL return to the untrusted code followed by an OCALL return into the initiator enclave.
12. The initiator enclave uses the key exchange library to process the 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.

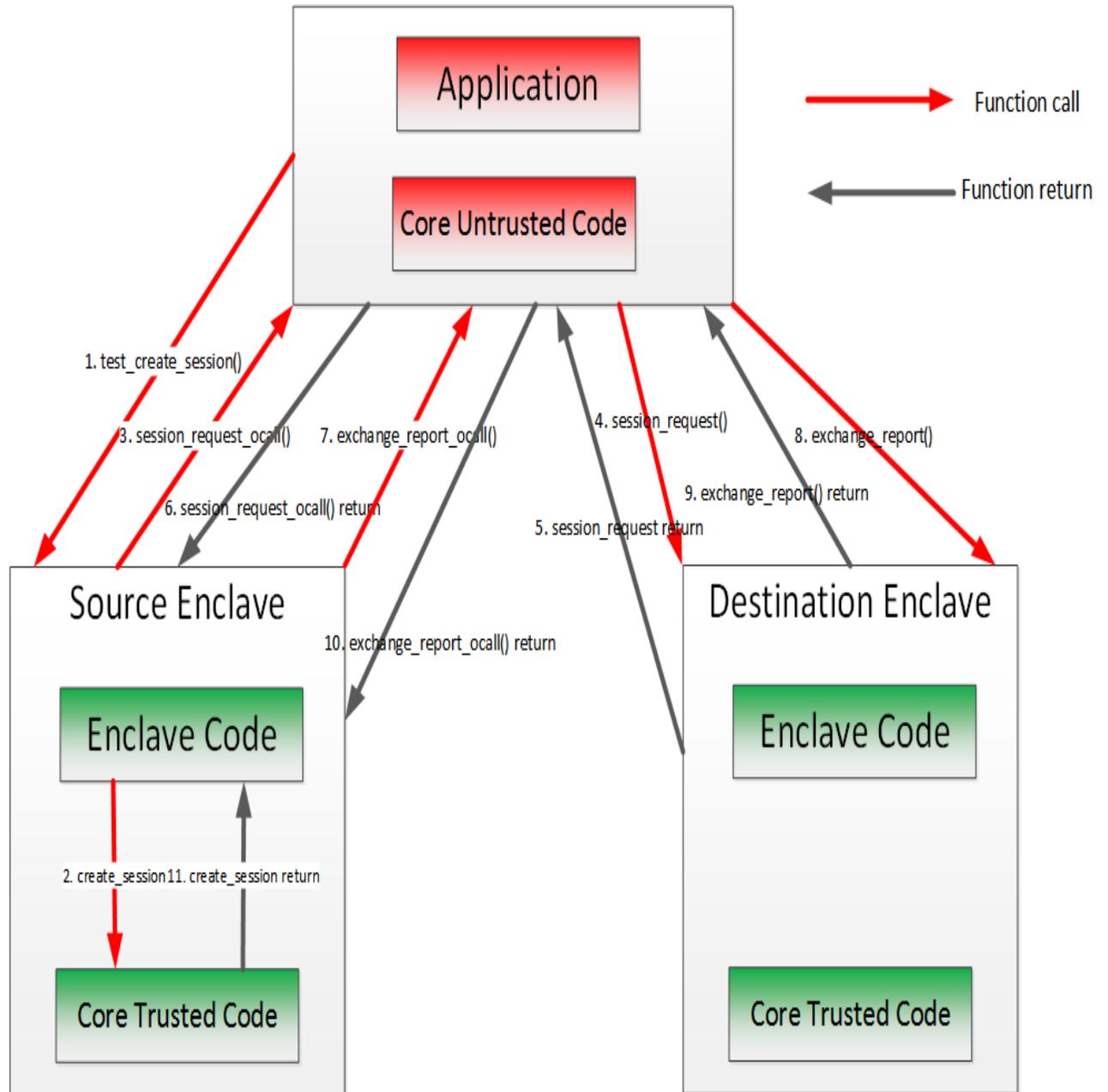


Figure 20 Secure Channel Establishment Flow with the DH Key Exchange Library

### 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 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.

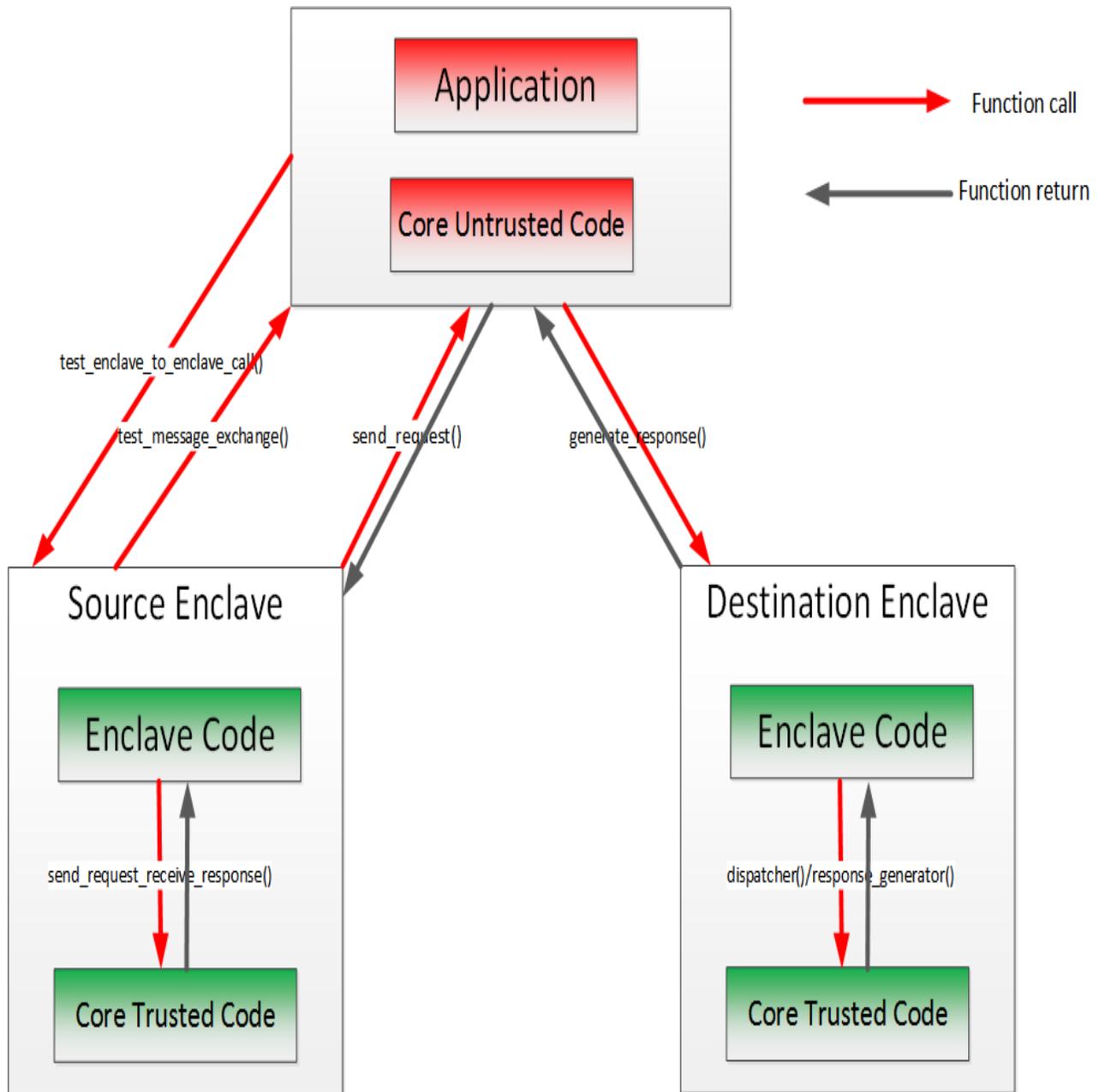


Figure 21 Secret Message Exchange Flow with the DH Key Exchange Library

### 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 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 `[Intel SGX SDK Install Path]src\RemoteAttestation`.

---

**NOTE:**

To run the sample code in the hardware mode, you need to access to Internet.

---

**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, Intel® Enhanced Privacy ID (Intel® 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 Intel® 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.

---

#### **NOTE:**

If you are unable to use either of the two pre-built untrusted key exchange static libraries, the source code for a sample untrusted key exchange library is included in the `isv_app` subfolder of the Remote Attestation sample application that is shipped with this SDK.

---

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.

---

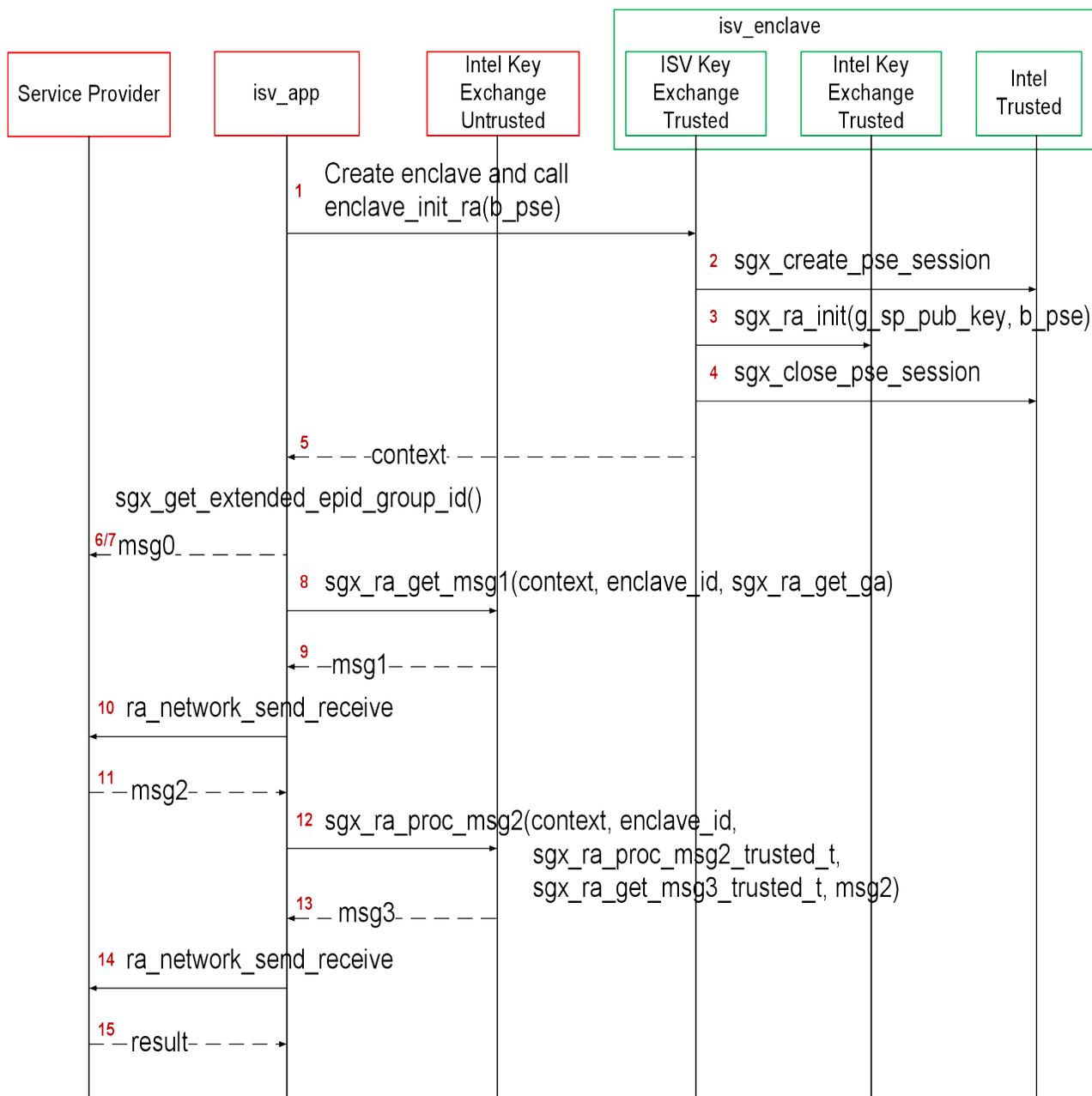


Figure 22 Remote Attestation and Trust Channel Establishment Flow

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 application calls `sgx_get_extended_epid_group_id()` and sends the value returned in `p_extended_epid_group_id` to the server in `msg0`.
7. The server checks whether the extended Intel® EPID group ID is supported. If the ID is not supported, the server aborts remote attestation.

---

**NOTE:**

Currently, the only valid extended Intel® EPID group ID is zero. The server should verify this value is zero. If the Intel® EPID group ID is not zero, the server aborts remote attestation.

---

8. The application 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.
9. `sgx_ra_get_msg1()` builds an S1 message = (ga || GID) and returns it to the app.
10. 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.
11. Application eventually receives  $S2 = gb \parallel SPID \parallel 2\text{-byte TYPE} \parallel 2\text{-byte KDF-ID} \parallel \text{SigSP}(gb, ga) \parallel \text{CMAC}_{SMK}(gb$

```
|| SPID || 2-byte TYPE || 2-byte KDF-ID || SigSP(gb,
ga)) || SigRL.
```

12. The application calls `sgx_ra_proc_msg2()`, passing in `S2` and the context.
13. The code in `sgx_ra_proc_msg2()` builds  $S3 = \text{CMAC}_{\text{SMK}}(M) || M$  where  $M = \text{ga} || \text{PS\_SECURITY\_PROPERTY} || \text{QUOTE}$  and returns it. Platform Services Security Information is included only if the app/enclave uses Platform Services.
14. Application sends the `msg3` to the SP by `ra_network_send_receive()`, and the SP verifies the `msg3`.
15. 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.

#### Remote Attestation with a Custom Key Derivation Function (KDF)

By default, the platform software uses the KDF described in the definition of the `sgx_ra_get_keys` API when the `sgx_ra_init` API is used to generate the remote attestation context. If the ISV needs to use a different KDF than the default KDF used by Intel® SGX PSW, the ISV can use the `sgx_ra_init_ex` API to provide a callback function to generate the remote attestation keys used in the SIGMA protocol (SMK) and returned by the API `sgx_ra_get_keys` (SK, MK, and VK). The decision to use a different KDF is a policy of the ISV, but it should be approved by the ISV's security process.

#### 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.

---

### **Using a Different Extended Intel® EPID Group for Remote Attestation**

The Intel® SGX platform software can generate Quotes signed by keys belonging to a more than one extended Intel® EPID Group. Before remote attestation starts, the ISV Service provider (SP) needs to know which extended Intel® EPID Group the PSW supports. The ISV SP will use this information to request Quote generation and verification in the correct extended Intel® EPID Group. The API `sgx_get_extended_epid_group_id` returns the extended Intel® EPID Group ID. The ISV application should query the currently configured extended Intel® EPID Group ID from the platform software using this API and sending it to the ISV SP. The ISV SP then knows which extended Intel® EPID Group to use for remote attestation. If the ISV SP does not support the

provided extended Intel® EPID Group, it will terminate the remote attestation attempt.

### ECDSA Remote Attestation

The Intel® SGX platform software consume The Intel® SGX Data Center Attestation Primitives (Intel® SGX DCAP) in order to support ECDSA attestation. The platform which creates the ECDSA attestation must support Flexible Launch Control (FLC).

The ECDSA Attestation key is created and owned by the owner of the remote attestation infrastructure but is certified by an Intel rooted key whose certificate is distributed by Intel. The Intel rooted certificate proves that the platform running the Intel® SGX enclave is valid and in good standing.

The application calls `sgx_select_att_key_id` to select the ECDSA attestation key from a list provided by the off-platform Quote verifier.

### 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 SDK in `[Intel SGX SDK Install Path]src\SealedData` folder. To compile, you only need to open the project with Microsoft\* Visual Studio 2019.

---

#### **NOTE:**

To run this sample in the hardware mode, access to the Internet.

---

#### **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.

### Intel® SGX2 Enclave

The project *Sgx2Enclave* illustrates creating and using the Intel® SGX 2.0 Enclave Dynamic Memory Management (Intel® SGX EDMM) enclave for the Intel® SGX 2.0 hardware platform, leveraging the `sgx_tedmm` library provided by the Intel® SGX SDK.

The project shows how to opt-in to Intel® SGX 2.0 behavior by linking the `sgx_tedmm` library and importing the `sgx_tedmm.edl` file. It also demonstrates Intel® SGX 2.0 API usage.

The code is shipped with the Intel® SGX SDK and is located in `[Intel SGX SDK Install Path]src\Sgx2Enclave`. You must open the project in Microsoft\* Visual Studio 2019.

---

**NOTE:**

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

---

### Switchless

The Switchless sample is designed to illustrate the usage and potential performance benefits of the Intel® Software Guard Extensions (Intel® SGX) Switchless Calls. It demonstrates the usage of `sgx_create_enclave_ex` and the Switchless Calls configuration. The Switchless sample EDL defines regular and switchless ECALLs and regular and switchless OCALLs. The sample application calls sample ECALLs/OCALLs in loops and compares the execution time of the regular and switchless calls.

### Protected Code Loader

Comparing the sample code in folder `SampleEnclavePCL` to the sample code in folder `SampleEnclave` demonstrates how to integrate the Intel® Software Guard Extensions Protected Code Loader (Intel® SGX PCL) into an ISV existing Intel® SGX project.

### Universal Windows Platform (UWP) Sample Enclave

Project `SampleEnclaveUwp` is a sample UWP application that uses the Intel® Software Guard Extensions (SGX) SDK.

This project is a port of the `SampleEnclave` project that comes with the Intel® SGX SDK to the UWP, based on the `BlankAppUWP C++` project generated by the Visual Studio Wizard. To use this project, you should be familiar with the `SampleEnclave Desktop` or `Win32` version.

The code is shipped with the Intel® SGX SDK and is located in `[Intel SGX SDK Install Path]src\SampleEnclaveUwp`. Open the project in the Microsoft\* Visual Studio 2019.

---

**NOTE:**

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

---

To debug this sample, see [Attaching to and Debugging an Enclave inside a Running UWP application](#)

## SampleSGXEnable

Project *SampleSGXEnable* is a sample application which is executed outside Enclave. It's main functions are:

1. Query Intel® SGX device's status.
2. Enable Intel® SGX device if it's set as *Software Control* inside BIOS.

The code is shipped with the Intel® SGX SDK and is located in `[Intel SGX SDK Install Path]src\SampleSGXEnable`. Open the project in the Microsoft\* Visual Studio 2019.

---

### **NOTE:**

This sample codes should be used inside APP installer to query and enable Intel® SGX device during installation.

---

## Sample Enclave for SGX MbedCrypto Library

The project shows you how to use SGX Mbedtls Cryptography Library within an enclave. This sample demonstrates the typical SGX Cryptography APIs' usages.

## Library Functions and Type Reference

This topic includes the following sub-topics to describe library functions and type reference for Intel® Software Guard Extensions 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 create or destroy enclaves:

- [sgx\\_create\\_enclave](#)
- [sgx\\_create\\_enclave\\_ex](#)
- [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 the Intel® Software Guard Extensions environment.

---

**NOTE:**

To run these functions in the hardware mode, you need to access to the Internet. Configure the system network proxy settings if needed.

---

**NOTE:**

Under certain conditions, the Architectural Enclave Service Manager (AESM) may fail to load Intel® signed enclaves due to the lack of resources. In this case,

---

---

`sgx_ra_get_msg1`, and `sgx_ra_proc_msg2` might return errors. To fix this problem, exit the applications with the loaded enclave to release the resource and retry.

---

These functions perform Intel® EPID quoting.

- [sgx\\_init\\_quote](#)
- [sgx\\_calc\\_quote\\_size](#)
- [sgx\\_get\\_quote\\_size](#)
- [sgx\\_get\\_quote](#)
- [sgx\\_report\\_attestation\\_status](#)
- [sgx\\_check\\_update\\_status](#)

These functions perform Intel® EPID quoting and ECDSA quoting.

- [sgx\\_select\\_att\\_key\\_id](#)
- [sgx\\_init\\_quote\\_ex](#)
- [sgx\\_get\\_quote\\_size\\_ex](#)
- [sgx\\_get\\_quote\\_ex](#)
- [sgx\\_get\\_supported\\_att\\_key\\_id\\_num](#)
- [sgx\\_get\\_supported\\_att\\_key\\_ids](#)

### Untrusted 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.

---

#### **NOTE:**

To run these functions in the hardware mode, you need to access to the Internet. Configure the system network proxy settings if needed.

---

---

#### **NOTE:**

Under certain conditions, the Architectural Enclave Service Manager (AESM) may fail to load Intel® signed enclaves due to the lack of resources. In this case, `sgx_ra_get_msg1`, and `sgx_ra_proc_msg2` might return errors. To fix this problem, exit applications with the loaded enclave to release the resource and retry.

---

These functions perform Intel® EPID attestation.

- [sgx\\_ra\\_get\\_msg1](#)
- [sgx\\_ra\\_proc\\_msg2](#)
- [sgx\\_get\\_extended\\_epid\\_group\\_id](#)

These functions perform Intel® EPID attestation and ECDSA attestation.

- [sgx\\_ra\\_get\\_msg1\\_ex](#)
- [sgx\\_ra\\_proc\\_msg2\\_ex](#)

### Untrusted Platform Service Function

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

---

**NOTE:**

To run this function in the hardware mode, you need to access to Internet. Configure the system network proxy settings if needed.

---

**NOTE:**

In some rare condition, some Intel® SGX signed enclaves will not be loaded because of out of resource. In such case, `sgx_ra_get_msg1`, and `sgx_ra_proc_msg2` might return errors. To fix this problem, exit applications with the loaded enclave to release the resource and retry.

---

- [sgx\\_get\\_ps\\_cap](#)

### Intel® SGX Enabling and Launch Control Functions

The enabling and launch control function helps you to enable the Intel® Software Guard Extensions (Intel® SGX) device and return appropriate status.

- [sgx\\_enable\\_device](#)
- [sgx\\_cap\\_enable\\_device](#)

This function provides an Enclave Signing Key Allow List Certificate Chain, which 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 Allow List Certificate Chain, you can provide the chain to the system by setting `sgx_register_wl_cert_chain`. Use `sgx_get_whitelist_size` to get the size of the current Enclave Signing Key Allow List Certificate Chain. Use `sgx_get_whitelist` to get the chain.

- [sgx\\_register\\_wl\\_cert\\_chain](#)
- [sgx\\_get\\_whitelist\\_size](#)
- [sgx\\_get\\_whitelist](#)

### Intel® SGX device capability Functions

The Intel® SGX device capability functions help you query the Intel SGX device status and the version of the PSW installed.

- [sgx\\_is\\_capable](#)
- [sgx\\_cap\\_get\\_status](#)
- [sgx\\_cap\\_get\\_psw\\_version\\_string](#)

## Trusted Libraries

The trusted libraries are static libraries that linked with the enclave binary. The Intel® Software Guard Extensions 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.

Trusted libraries built for HW mode (for example, not for simulation) contain a string with the release number. The string version, which uses the library name as prefix, is defined when the SDK is built and consists of various parameters such as the product number, SVN revision number, build number, and so on. This mechanism ensures all trusted libraries shipped in a given SDK release will have the same version number and allows quick identification of the trusted libraries linked into an enclave.

For instance, `sgx_tstdc.lib` contains a string version like `SGX_TSTDC_VERSION_1.0.0.0`. Of course, the last digits vary depending on the SDK release.

---

### **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 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](#)
- [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 SDK provides an API to allow you to register functions, or exception handlers, to handle a limited set of hardware exceptions. When one of the enclave supported hardware exceptions occurs within the enclave, the registered exception handlers will be called in a specific order until an exception handler reports that it has handled the exception. For example, issuing a CPUID instruction inside an Enclave will result in a #UD fault (Invalid Opcode Exception). ISV enclave code can call `sgx_register_exception_handler` to register a function of type `sgx_exception_handler_t` to respond to this exception. Any custom exception handlers should check the validity of the exception, as it may be coming from an untrusted source. To check a list of enclave supported exceptions, see [Intel® Software Guard Extensions Programming Reference](#).

---

#### **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 handler.

---

#### **NOTE:**

Custom exception handling only saves general purpose registers in `sgx_exception_info_t`. You should be careful when touching other registers in the exception handlers.

---

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 using the CPUID information within an enclave, then the enclave code must make an OCALL to perform the CPUID instruction in the untrusted application. The Intel® SGX SDK provides two functions in the library `sgx_tstdc` to obtain CPUID information through an OCALL:

- `sgx_cpuid`
- `sgx_cpuid_ex`

In addition, the Intel SGX SDK also provides the following intrinsics which call the above functions to obtain CPUID data:

- `__cpuid`
- `__cpuidex`

Both the functions and intrinsics result in an OCALL to the uRTS library to obtain CPUID data. The results are returned from an untrusted component in the system. It is recommended that threat evaluation be performed to ensure that 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 to handle the exception generated from issuing the CPUID instruction (unless the third party library registers its own exception handler for 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:

1. Retrieve the *cached* CPUID result and populate the CPUID instruction output registers.
2. Advance the RIP to bypass the CPUID instruction and complete the exception handling.

### 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>`:

1. Add `$(VC_VC_IncludePath)` to Include Directories.
  2. 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 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](#)
- [Trusted Platform Service Functions](#)
- [Diffie–Hellman \(DH\) Session Establishment Functions](#)
- [Custom Alignment Interfaces](#)

### Intel® Software Guard Extensions Instruction Wrapper Functions

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

- [sgx\\_get\\_key](#)
- [sgx\\_create\\_report](#)
- [sgx\\_verify\\_report](#)

The `sgx_tservice` library also provides two help functions:

- [sgx\\_self\\_report](#) for getting a self cryptographic report
- [sgx\\_self\\_target](#) for getting a self target info of an enclave.

### Intel® Software Guard Extensions Sealing and Unsealing Functions

The `sgx_tservice` library provides the following functions:

- Exposes APIs to create sealed data which is both confidentiality and integrity protected.
- Exposes an API to unseal sealed data inside the enclave.
- Provides APIs to authenticate and verify the input data with AES-GMAC.

See the following related topics for more information.

- [sgx\\_seal\\_data](#)
- [sgx\\_seal\\_data\\_ex](#)
- [sgx\\_unseal\\_data](#)
- [sgx\\_mac\\_aadata](#)
- [sgx\\_mac\\_aadata\\_ex](#)
- [sgx\\_unmac\\_aadata](#)

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](#)

### SealLibrary 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 Intel® Advanced Encryption Standard New Instructions (Intel® 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.

### Trusted 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.

---

#### **NOTE:**

To run these functions in the hardware mode, you need to access to Internet. Configure the system network proxy settings if needed.

---

- [sgx\\_create\\_pse\\_session](#)
- [sgx\\_close\\_pse\\_session](#)
- [sgx\\_get\\_ps\\_sec\\_prop](#)
- [sgx\\_get\\_ps\\_sec\\_prop\\_ex](#)
- [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

The `sgx_tservice` library provides the following functions to 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](#)

### Custom Alignment Interfaces

The `sgx_tservice` library provides a set of interfaces that facilitate custom alignment of secrets and structures that contain secrets. Different interfaces are used for secrets that are statically-defined (stack, global, static) versus dynamically-defined (heap) and for secrets in C++ code versus C code.

See the following related topics for more information:

- class template [custom\\_alignment\\_aligned](#)
- [sgx\\_get\\_aligned\\_ptr](#)
- [sgx\\_aligned\\_malloc](#)
- [sgx\\_aligned\\_free](#)

### C Standard Library

The Intel® Software Guard Extensions 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 `[Intel SGX SDK Install Path]include\tlibc`.

`sgx_tstdc` provides a subset of C11 functions that are ported from the OpenBSD\* project. Unsupported functions are not allowed inside an enclave for the 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 C11 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 a 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 use it 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 SDK already supports event, 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). If you use any of the APIs below, you must first import the needed OCALL functions from `sgx_tstdc.edl`. Otherwise, you 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 Intel® SGX Thread Synchronization library supports, as well as the OCALLs that each API function needs.

	Function API	OCall Function
Mutex Synchronization	<a href="#">sgx_thread_mutex_init</a>	
	<a href="#">sgx_thread_mutex_destroy</a>	
	<a href="#">sgx_thread_mutex_lock</a>	<code>sgx_thread_wait_untrusted_event_ocall</code>
	<a href="#">sgx_thread_mutex_trylock</a>	
	<a href="#">sgx_thread_mutex_unlock</a>	<code>sgx_thread_set_untrusted_event_ocall</code>
Condition Variable Synchronization	<a href="#">sgx_thread_cond_init</a>	
	<a href="#">sgx_thread_cond_destroy</a>	
	<a href="#">sgx_thread_cond_wait</a>	<code>sgx_thread_wait_untrusted_event_ocall</code> <code>sgx_thread_setwait_untrusted_events_ocall</code>
	<a href="#">sgx_thread_cond_signal</a>	<code>sgx_thread_set_untrusted_event_ocall</code>
	<a href="#">sgx_thread_cond_broadcast</a>	<code>sgx_thread_set_multiple_untrusted_events_ocall</code>
	Event Synchronization	<a href="#">sgx_create_event</a>

	<a href="#">sgx_set_event</a>	sgx_thread_wait_untrusted_event_ocall sgx_thread_set_untrusted_event_ocall sgx_thread_set_untrusted_event_ocall sgx_thread_set_multiple_untrusted_events_ocall
	<a href="#">sgx_reset_event</a>	sgx_thread_wait_untrusted_event_ocall sgx_thread_set_untrusted_event_ocall
	<a href="#">sgx_destroy_event</a>	sgx_thread_wait_untrusted_event_ocall sgx_thread_set_untrusted_event_ocall
	<a href="#">sgx_wait_for_single_object</a>	sgx_thread_wait_untrusted_event_ocall sgx_thread_set_untrusted_event_ocall sgx_thread_wait_untrusted_event_ocall sgx_thread_setwait_untrusted_events_ocall
	<a href="#">sgx_wait_for_multiple_objects</a>	sgx_thread_wait_untrusted_event_ocall sgx_thread_set_untrusted_event_ocall sgx_thread_wait_untrusted_event_ocall sgx_thread_setwait_untrusted_events_ocall
Thread Management	<a href="#">sgx_thread_self</a>	
	<a href="#">sgx_thread_equal</a>	

### Query CPUID inside Enclave

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

- `sgx_cpuid`
- `sgx_cpuidex`

### Secure Functions

The Intel® Software Guard Extensions SDK provides some secure functions. An enclave project can include `<mbusafect.h>` to use them.

See [Supported C Secure Functions](#) for a list of supported secure functions definitions within an enclave.

### Non-Local Jumps

The C standard library provides a pair of functions, `setjmp` and `longjmp`, that can be used to perform non-local jumps. `setjmp` saves the current program state into a data structure. `longjmp` can later use this data structure to restore the execution context. This means that after `longjmp`, execution continues at the `setjmp` call site.

Since `setjmp/longjmp` may transfer execution from one function to a pre-determined location in another function, normal stack unwinding does not occur. As a result, you must use this functionality carefully, ensuring that an enclave only calls `setjmp` in a valid context. You should also perform extensive security validation to ascertain that the enclave never uses these functions in such a way it could result in undefined behavior. Typical use of `setjmp/longjmp` is the implementation of an exception mechanism (error handling). However, you must never use these functions in C++ programs. You should use the standard CEH instead. You are recommended to review the information provided at [cert.org](http://cert.org) on how to use `setjmp/longjmp` securely.

### C++ Language Support

The Intel® Software Guard Extensions 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: C++ standard is chosen as the baseline for the Intel® Software Guard Extensions 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, you are recommended 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 inside the enclave 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 SDK includes a trusted version of the C++ standard library that conforms to the C++11 standard. Visual Studio\* uses `sgx_tcxx.lib`, which has been ported from `libc++` and supports most C++11 features.

As for the C++ standard library, most functions will work just as its untrusted counterpart, but here is a high level summary of features that are not supported inside the enclave:

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

However, only 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.

### **Known Issue for C++ Exception**

There is a known issue for C++ exception handling inside the enclave. An exception object cannot be rethrown after it has been handled once (after a catch block), even though the object was not expected to be destroyed. The issue exists only in the 32-bit enclave.

Here is an example:

```
void go2 ()
```

```

{
    try {
        throw 1;
    }
    catch (...) {
        try {
            throw;
        }
        catch (...) {
            } // exception object was destroyed unexpectedly
            throw; // Crash!
        }
    }
}

```

### Cryptography Library

The Intel® Software Guard Extensions Software Development Kit (Intel® SGX SDK) includes a trusted cryptography library named `sgx_tcrypto`. It contains cryptographic functions used by other trusted libraries included in the SDK, such as the `sgx_tservice` library. Thus, functionality of this library is limited.

- `sgx_sha256_msg`
- `sgx_sha256_init`
- `sgx_sha256_update`
- `sgx_sha256_get_hash`
- `sgx_sha256_close`
- `sgx_rijndael128GCM_encrypt`
- `sgx_rijndael128GCM_decrypt`
- `sgx_aes_gcm128_enc_init`
- `sgx_aes_gcm128_enc_update`
- `sgx_aes_gcm128_enc_get_mac`
- `sgx_aes_gcm_close`
- `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`
- `sgx_ecdsa_verify_hash`
- `sgx_rsa3072_sign`
- `sgx_rsa3072_sign_ex`
- `sgx_rsa3072_verify`
- `sgx_create_rsa_key_pair`
- `sgx_create_rsa_priv1_key`
- `sgx_create_rsa_priv2_key`
- `sgx_create_rsa_pub1_key`
- `sgx_free_rsa_key`
- `sgx_rsa_priv_decrypt_sha256`
- `sgx_rsa_pub_encrypt_sha256`
- `sgx_calculate_ecdsa_priv_key`
- `sgx_ecc256_calculate_pub_from_priv`
- `sgx_hmac_sha256_msg`
- `sgx_hmac256_init`
- `sgx_hmac256_update`
- `sgx_hmac256_final`
- `sgx_hmac256_close`
- `sgx_sha384_msg`
- `sgx_sha384_init`
- `sgx_sha384_update`
- `sgx_sha384_get_hash`
- `sgx_sha384_close`

If you need additional cryptographic functionality, you can use the Intel® Integrated Performance Primitives (Intel® IPP) Cryptography library. The Intel® IPP Cryptography 2021 Update 3 is linked to `sgx_tcrypto.lib` and its API is directly accessible. The Intel® IPP Cryptographic header files are located in `[Intel SGX SDK Install Path]include\ipp`. The Intel® IPP Cryptographic library is included in the SDK under the [Community Licensing](#).

The trusted cryptography library is based on an underlying general-purpose cryptographic library: Intel® Integrated Performance Primitives (Intel® IPP) Cryptography library or Intel® Software Guard Extensions SSL cryptographic library (Intel® SGX SSL).

The default build uses precompiled, optimized libraries, which include Intel® IPP libraries. In addition, the Intel® IPP Cryptographic header files are located in `[Intel SGX SDK Install Path]include/ipp`. If you want to use Intel® SGX SSL instead of Intel® IPP you should use the non-optimized code implementation. Follow the README.md for detailed instructions.

If you need additional cryptographic functionality, you can use the general-purpose cryptographic library, and its corresponding header files. The underlying trusted libraries are linked into `libsgx_tcrypto.a`.

Directly accessing Intel® SGX SSL API is possible after updating the Enclave EDL and the Application project with the requirements described in [Intel® SGX SSL Developer Guide](#), section 2.2. Using Intel® SGX SSL Library.

---

### **NOTE**

To get internal OpenSSL\* error codes, you need to build `sgx_tcrypto` in DEBUG mode and declare `extern unsigned long openssl_last_err`, which holds OpenSSL error code upon failure.

See more information at the [Intel® Software Guard Extensions SSL cryptographic library GitHub\\* repository](#).

Known limitations:

- Intel SGX SSL library registers a CPUID exception handler which handles CPUID exceptions on certain leaves:
  - leaf 0x0
  - leaf 0x1
  - leaf 0x4, sub leaf 0x0
  - leaf 0x7, sub leaf 0x0
- Enclaves using one of the mentioned leaves may have a different behavior upon moving from Intel IPP to Intel SGX SSL.
- When running `sgx-gdb` on an enclave built with Intel SGX SSL, several SIGILL signals might be raised (since OpenSSL code has some cpuid instruction calls), `gdb continue` will continue executing your program.

---

### **NOTE**

See more information on cryptography in [Intel® Integrated Performance Primitives Cryptography Developer Reference](#).

---

### Trusted 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\\_init\\_ex](#)
- [sgx\\_ra\\_get\\_keys](#)
- [sgx\\_ra\\_close](#)

### Intel® Protected File System Library

Intel® Protected File System Library provides protected files API for Intel® SGX enclaves. It supports a basic subset of the regular C file API and enables you to create files and work with them as you would normally do from a regular application.

With this API, the files are encrypted and saved on the untrusted disk during a write operation, and they are verified for confidentiality and integrity during a read operation.

To encrypt a file, you should provide a file encryption key. This key is a 128 bits key, and is used as a key derivation key, to generate multiple encryption keys. According to “NIST Special Publication 800-108 - Recommendation for Key Derivation Using Pseudorandom Functions”: “The key that is input to a key derivation function is called a key derivation key. To comply with this Recommendation, a key derivation key shall be a cryptographic key (see Section 3.1). The key derivation key used as an input to one of the key derivation functions specified in this Recommendation can be generated by an approved cryptographic random bit generator (e.g., by a deterministic random bit generator of the type specified in [5]), or by an approved automated key-establishment process (e.g., as defined in [1] and [2])”. For more details, please refer to [NIST SP 800-108](#) document.

Another option is to use automatic keys derived from the enclave sealing key (see disadvantages of this approach in the topic Using the Protected FS Automatic Keys API).

- [sgx\\_fopen](#)
- [sgx\\_fopen\\_auto\\_key](#)
- [sgx\\_fclose](#)
- [sgx\\_fread](#)
- [sgx\\_fwrite](#)

- `sgx_fflush`
- `sgx_ftell`
- `sgx_fseek`
- `sgx_feof`
- `sgx_ferror`
- `sgx_clearerr`
- `sgx_remove`
- `sgx_fexport_auto_key`
- `sgx_fimport_auto_key`
- `sgx_fclear_cache`

#### Protected FS Usage Limitation

Since the Protected Files have meta-data embedded in them, only one file handle can be opened for writing at a time, or many file handles for reading. OS protection mechanism is used for protecting against accidentally opening more than one 'write' file handle. If this protection is bypassed, the file will get corrupted. An open file handle can be used by many threads inside the same enclave, the APIs include internal locks for handling this and the operations will be executed by one.

#### Protected FS Error Codes

The Protected File System (FS) API tries to preserve the original C file API errors. A local `errno` (enclave internal) is also set for APIs that require this according to the original C file API documentation.

When the Protected FS API is accessing the OS file system, if an error is returned, that error will be provided back to the caller of the Protected FS API. In addition, when possible, it returns an `EXXX` error code for internal errors (for example, the `EACCES` error code is returned when trying to write a file that was opened as read-only, or `ENOMEM` is returned when an internal attempt to allocate memory fails). Several special error codes were added, like `SGX_ERROR_FILE_NAME_MISMATCH`, for the cases when the current file name does not match the internal file name. You can find these error codes and their explanations in `sgx_error.h`.

#### Protected FS Application Layout

The following figure demonstrates how the Protected File System (FS) works inside an Intel® SGX application:

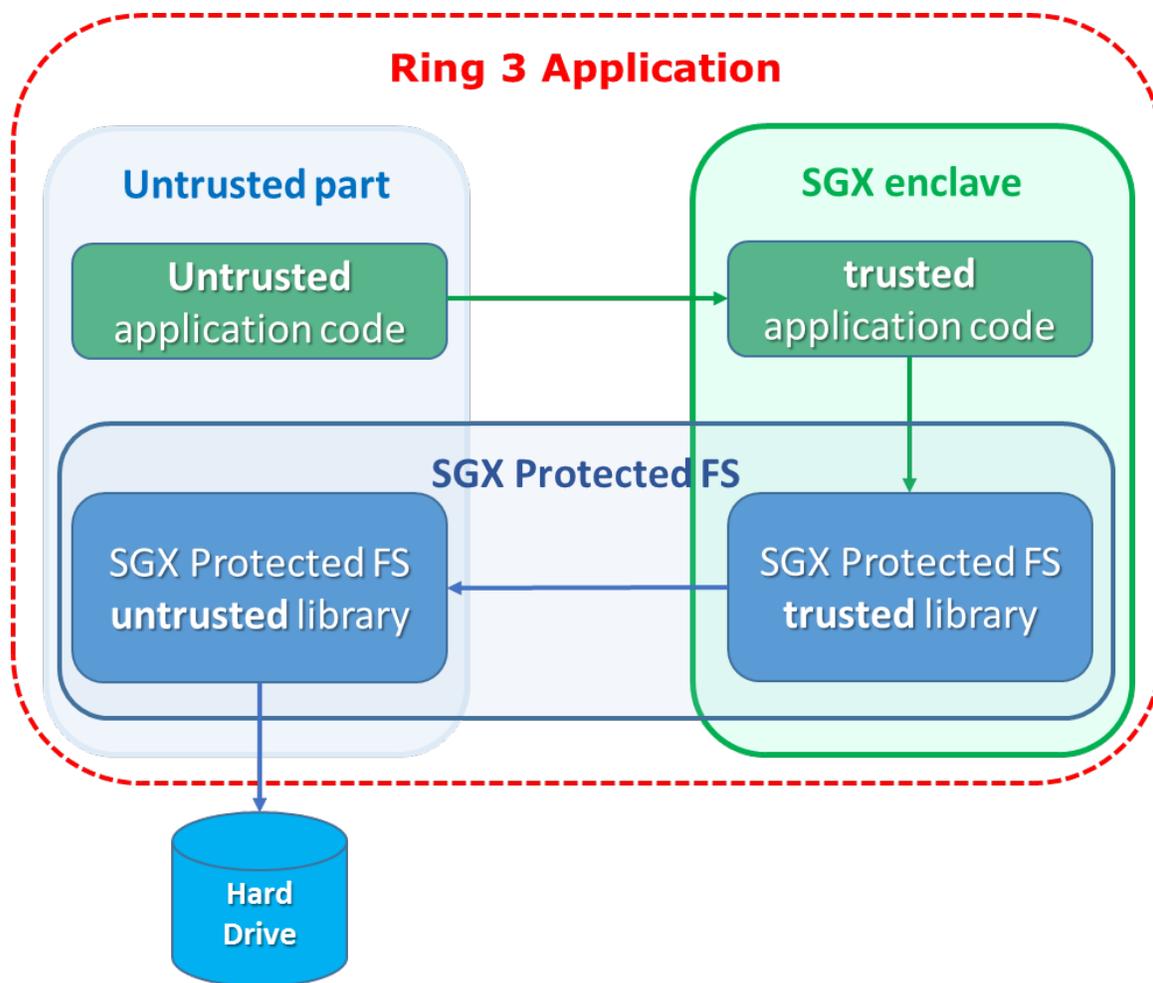


Figure 23 Protected File System Layout

To use the Intel SGX Protected File System libraries:

1. The enclave must be linked with `sgx_tprotected_fs.lib`
2. The application must be linked with `sgx_uprotected_fs.lib`
3. The enclave's EDL file must 'import' all the functions from `sgx_tprotected_fs.edl`
4. The source files should 'include' `sgx_tprotected_fs.h`

#### Protected FS S3/S4 Important Note

To enhance the performance of the Protected File System, cache is used to save the user's data inside the enclave, and only when the cache is full it is flushed to disk.

In case of S3/S4 transitions, data in the cache will be lost (the entire enclave is lost). Therefore, all file handles must be flushed or closed before entering S3/S4. If no action is taken, the file integrity is not harmed, only the latest written data that was not flushed will be lost.

#### Using the Protected FS Automatic Keys API

Automatic keys are derived from the enclave sealing key (with MRSIGNER), so the files are bound to all the enclaves signed by the same signer on this particular machine. To transfer a file from one machine to another, you should follow the key export and import procedure below. There are several cases when using automatic keys is not recommended.

- Disaster recovery – trying to open files created on one machine on a different machine will fail (unless you follow the procedure described below). Therefore, automatic disaster recovery may not work properly.
- VM migrations – currently, Intel® SGX does not support automatic enclave keys transfer in VM migration. Therefore, enclaves running on servers that use VM migrations cannot use the auto key API.

#### File Transfer with the Automatic Keys API

For files that were created with `sgx_fopen_auto_key`, to transfer a file from one enclave to another, or to another enclave on a different machine, follow this procedure:

1. Close all open handles to the file.
2. Call the `sgx_fexport_auto_key` API. This API returns the last encryption key that was used to encrypt the meta-data node of the file.
3. Transfer the file to the destination enclave, and provide the key in a safe method to that enclave.
4. Call the `sgx_fimport_auto_key` API. This API will re-encrypt the meta-data node with a new encryption key, derived from the local enclave seal key.
5. Open the file with `sgx_fopen_auto_key` as usual.

#### Protected FS Security Non-Objectives

In order to mitigate file swapping attacks (with two valid files), file names are checked during a file open operation (verifies that the current name is equal to the file name the file was created with). However, if two files are created with the same file name, there is no way to protect against such a swapping attack.

Since the files are saved in the regular FS, there is no protection against malicious file deletion or modification – this will only be detected when trying to read or write the modified section from the file (decryption will fail).

There are several things that are not protected when using the Protected FS API, and anyone who can access the OS can see them:

1. File name
2. File size (up to 4KB granularity)
3. File modification date
4. Key type (user or auto)
5. Usage patterns
6. Read/Write offsets

If any of those items might expose sensitive information, and help a potential attacker, the enclave developer should add defense mechanisms on their own to protect against this. For example, an implementation of a “secure browser” should not save the ‘cookies’ with the names of their related websites, because an attacker can learn from that the user’s browsing history.

### **TCMalloc Library**

The Intel® Software Guard Extensions SDK includes a trusted version of the TCMalloc library. The library is named `sgx_tcmalloc`, and can only be used inside an enclave. `sgx_tcmalloc` provides high performance memory allocation and deallocation functions that are ported from gperftools-2.7:

- `malloc`
- `free`
- `realloc`
- `calloc`
- `memalign`

Do the following to enable TCMalloc in Intel® SGX:

1. Set the enclave project's `Global Heap Size` value equal or larger than `0xA00000`.

The `Global Heap Size` setting path:

```
enclave project->Intel® SGX Configuration->Enclave Settings->Global Heap Size
```

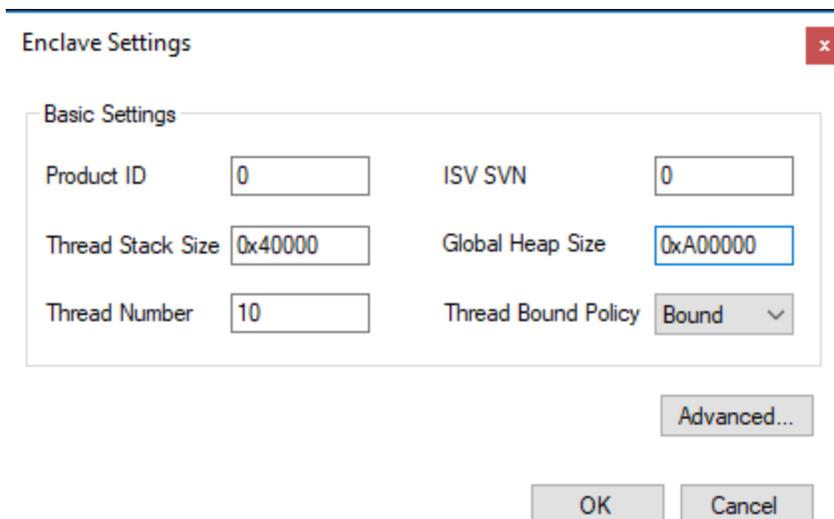


Figure 24 Global Heap Size

2. Add `sgx_tcmalloc.lib` into the target enclave project's `Additional Dependencies` option.

The `Additional Dependencies` setting path:

enclave project->Properties->Linker->Input->Additional Dependencies

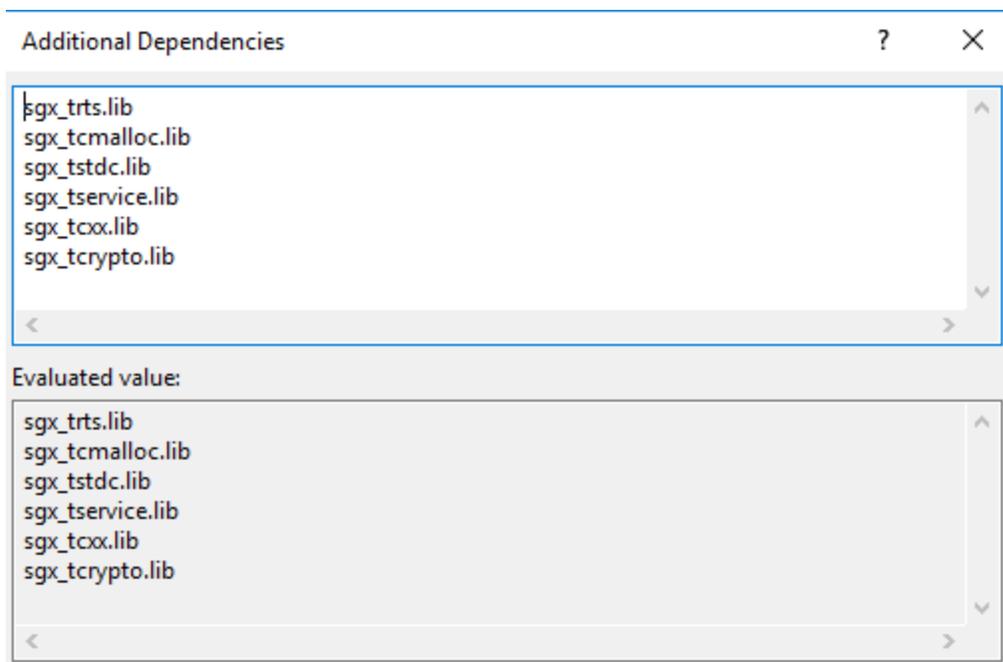


Figure 25 Additional Dependencies

---

**NOTE:**

The `sgx_tcmalloc.lib` must be added before `sgx_tstdc.lib`, or `dlmalloc` will be used.

---

**Enclave Dynamic Memory Management Library**

The Intel® SGX SDK includes a trusted library, `sgx_tedmm.lib`, that provides support for modifying the permissions of committed pages in an enclave. Intel® SGX EDMM behavior is only available on Intel® SGX 2.0 hardware platforms with 2.0 Platform Software and Intel® SGX 2.0-capable Windows\* OS.

An Intel® SGX EDMM-capable enclave opts-in to Intel® SGX 2.0 behavior by linking against the `sgx_tedmm` library and importing the file `sgx_tedmm.edl`. In addition to supporting explicit requests to modify page permissions, this enclave will automatically modify any RWX code pages to RX after relocation completes. An Intel® SGX EDMM-capable enclave will also execute on a system without support for Intel® SGX 2.0, but will not be able to perform Intel® SGX EDMM operations. The Intel® SGX EDMM APIs will report `SGX_ERROR_FEATURE_NOT_SUPPORTED` in this situation.

**Update Project Settings for Enclave Project**

To create an Intel® SGX EDMM-capable enclave, adjust your Enclave Project settings to add the Intel® SGX EDMM library.

**Linker->Input->Additional Dependencies:** Add `sgx_tedmm.lib`

**Import Intel® SGX EDMM EDL file**

The Intel® SGX EDMM EDL file should be imported by the enclave EDL file as shown in the example. Note including this edl file will expose an additional OCALL for the enclave.

```
enclave {  
    from "sgx_tedmm.edl" import *;  
  
    trusted {  
    };  
    untrusted {  
    };  
};
```

**EDMM Library API**

- `sgx_virtual_protect`

### Switchless Calls Library

The untrusted portion of the Switchless Calls is integrated into the uRTS library. The trusted part is provided by `sgx_tswitchless.lib` library. The trusted library does not expose any API functions. It just enables the Switchless Calls feature inside the enclave.

Developers can enable Switchless Calls using the 'transition\_using\_threads' keyword in the enclave EDL file .

At runtime, the enclave must be created using the `sgx_create_enclave_ex` API, providing a Switchless Calls configuration structure.

### Protected Code Loader Library

The untrusted portion of Intel® SGX PCL is integrated into the uRTS library. The trusted part is provided by `sgx_pcl.lib`. The trusted library does not expose any API to the ISV portion of the enclave. See 'Integrating Intel® SGX PCL with an existing Intel® SGX solution' above for a detailed description of how these libraries are used. Intel® SGX PCL is not supported in simulation mode.

### Intel® SGX Mbedtls Cryptography Library

The Intel® SGX Mbedtls Cryptography Library is a customized version of the Mbed TLS C library that provides cryptographic primitives, X.509 certificate manipulation and the support for SSL/TLS and DTLS protocols. Documentation for Mbed TLS can be found at [trustedfirmware.org](https://trustedfirmware.org). The Intel® Software Guard Extensions Software Development Kit (Intel® SGX SDK) includes a trusted version of Mbedtls cryptography library for C/C++, named `libsgx_mbedcrypto.a`.

Most functions of the Mbedtls cryptography standard library work similarly to their untrusted counterpart. However, some features are not supported within the enclave, including I/O related functions, functions that rely on a locale library, and any other functions that require system calls.

To use Intel® SGX Mbedtls Cryptography Library within an enclave, update the Makefile with the following steps:

1. Include '-I\$(SGX\_SDK)/include/mbedtls' in the enclave header files options.
2. Add '-lsgx\_mbedcrypto' to the enclave linking options.

To enable logging for the SGX MbedtIs Cryptography Library within the enclave, please define `printf`.

## 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

```
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. An enclave created in non-debug mode cannot be debugged.

1 indicates to create the enclave in debug mode. 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. In release builds, the value of `SGX_DEBUG_FLAG` is 0. In debug and pre-release builds, the value of `SGX_DEBUG_FLAG` is 1 by default.

### **launch\_token [in/out]**

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 attempts to create a valid `sgx_launch_token_t` object and store it in the buffer. You 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 attempts 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]**

Pointer to an `sgx_enclave_id_t` that receives the enclave ID or handle. Must not be NULL.

### **misc\_attr [out, optional]**

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 is 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 cannot be opened. Possible reasons: the enclave file is not found or you have no privilege to access the enclave file.

### **SGX\_ERROR\_INVALID\_METADATA**

The metadata embedded within the enclave image is corrupted 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. Possible reasons: you are creating too many enclaves, the enclave requires too much memory, or one of the Architecture Enclaves for this operation cannot be loaded.

### **SGX\_ERROR\_NO\_DEVICE**

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

### **SGX\_ERROR\_MEMORY\_MAP\_CONFLICT**

During the enclave creation, a race condition for mapping memory between the loader and another thread occurred. The loader may fail to map virtual address. Create the enclave again.

### **SGX\_ERROR\_DEVICE\_BUSY**

The Intel® SGX driver or a low level system is busy when creating the enclave. Create the enclave again.

### **SGX\_ERROR\_MODE\_INCOMPATIBLE**

The target enclave mode is incompatible with the mode of the current RTS. Possible reasons: 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\_HYPERV\_ENABLED**

Incompatible versions of Windows\* 10 OS and Hyper-V\* are detected. Disable Hyper-V on the target machine.

#### **SGX\_ERROR\_PCL\_ENCRYPTED**

Enclave is encrypted. This function cannot be used to load an enclave that was encrypted by the encryption tool. Use `sgx_create_enclave_ex`.

#### **SGX\_ERROR\_FEATURE\_NOT\_SUPPORTED**

Requested feature is not supported. Possible features are KSS, EDMM, and, if the application is a UWP App, TCS\_POLICY\_BIND.

#### **SGX\_ERROR\_UNEXPECTED**

Unexpected error is detected.

## Description

The `sgx_create_enclave` function loads and initializes the enclave using the enclave file name and a launch token. If the launch token is incorrect, the function gets a new one and saves it back to the input parameter “token”. In this case, the parameter “updated” indicates that the launch token is updated.

If both enclave and launch tokens are valid, the function returns 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 the Intel® SGX hardware, and it 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 simulation 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 OS may randomize or not randomize the base address. A compromised loader or operating system (both of which are outside the TCB) can remove the randomization entirely. The enclave writer should not rely on the randomization of the base address of the enclave.

Do not call the `sgx_create_enclave` function as part of initialization of a DLL. For instance, `sgx_create_enclave` hangs when it is called from the global object constructor of a C++ class in a DLL.

## Requirements

Header	<code>sgx_urts.h</code>
Library	<code>sgx_urts.lib</code> or <code>sgx_urts_sim.lib</code> (simulation)

### **`sgx_create_enclave_ex`**

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

Enables extended features, Intel® SGX PCL, Switchless Calls initialization, and Key Separation & Sharing (KSS).

`sgx_create_enclave_ex` is a macro for the `sgx_create_enclave_exa` (ANSI) or `sgx_create_enclave_exw` (Unicode) function.

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

## Syntax

```
sgx_status_t sgx_create_enclave_ex(
    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,
    const uint32_t ex_features,
    const void* ex_features_p[32]
);
```

## 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 the Unicode character set, `file_name` should be a 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 creating the enclave in a non-debug mode. An enclave created in a non-debug mode cannot be debugged.

1 indicates creating the enclave in a debug mode. The code or data memory inside an enclave created in a debug mode is accessible by a debugger or another software outside the enclave. Thus, this enclave is *not* under the same memory access protections as a non-debug enclave.

You should create enclaves in the debug mode for debug purposes only. To create a debuggable enclave, you can use a helper macro `SGX_DEBUG_FLAG`. In release builds, the value of `SGX_DEBUG_FLAG` is 0. In debug and pre-release builds, the value of `SGX_DEBUG_FLAG` is 1 by default.

### **launch\_token [in/out]**

Pointer to an `sgx_launch_token_t` object used to initialize the enclave to be created. Must not be NULL. You can provide an all-0 buffer as the `sgx_launch_token_t` object, in which case, the function attempts to create a valid `sgx_launch_token_t` object and store it in the buffer. You 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 attempts to update it to a valid one.

#### **launch\_token\_updated [out]**

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

#### **enclave\_id [out]**

Pointer to an [sgx\\_enclave\\_id\\_t](#) that receives the enclave ID or handle. Must not be NULL.

#### **misc\_attr [out, optional]**

Pointer to an [sgx\\_misc\\_attribute\\_t](#) structure that receives the misc select and attributes of the enclave. This pointer can be NULL if the information is not needed.

#### **ex\_features [in]**

Bitmask defining the extended features to activate on the enclave creation.

Bit [0] – enable the Intel® SGX PCL.

Bit [1] – enable Switchless Calls.

Bit [2] - enable Key Separation & Sharing (KSS).

Bits [3:31] – reserved, must be 0.

#### **ex\_features\_p [in]**

Array of pointers to extended feature control structures. The index of the extended feature control structure in the array is the same as the index of the feature enable bit in `ex_features`.

`ex_features_p[0]` – pointer to an Intel® SGX PCL sealed key.

`ex_features_p[1]` – pointer to the [sgx\\_uswitchless\\_config\\_t](#) structure.

`ex_features_p[2]` - pointer to the [sgx\\_kss\\_config\\_t](#) structure.

`ex_features_p[3:31]` – reserved, must be NULL.

#### **Return value**

#### **SGX\_SUCCESS**

Enclave is loaded and initialized successfully.

### **SGX\_ERROR\_INVALID\_ENCLAVE**

Enclave file is corrupted.

### **SGX\_ERROR\_INVALID\_PARAMETER**

'enclave\_id', 'updated', or 'token' parameter is NULL.

### **SGX\_ERROR\_OUT\_OF\_MEMORY**

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

### **SGX\_ERROR\_ENCLAVE\_FILE\_ACCESS**

Enclave file cannot be opened. Possible reasons: the file is not found or you have no privilege to access the file.

### **SGX\_ERROR\_INVALID\_METADATA**

Metadata embedded within the enclave image is corrupted or missing.

### **SGX\_ERROR\_INVALID\_VERSION**

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

### **SGX\_ERROR\_INVALID\_SIGNATURE**

Signature for the enclave is not valid.

### **SGX\_ERROR\_OUT\_OF\_EPC**

Protected memory has run out. Possible reasons: you are creating too many enclaves, the enclave requires too much memory, or one of the Architecture Enclaves for this operation cannot be loaded.

### **SGX\_ERROR\_NO\_DEVICE**

Intel® SGX device is not valid. Possible reasons: the Intel® SGX driver is disabled or not installed.

### **SGX\_ERROR\_MEMORY\_MAP\_CONFLICT**

During enclave creation, a race condition for mapping memory between the loader and another thread occurred. The loader may fail to map virtual address. Create the enclave again.

### **SGX\_ERROR\_DEVICE\_BUSY**

Intel®SGX driver or a low level system is busy when creating the enclave. Create the enclave again.

### **SGX\_ERROR\_MODE\_INCOMPATIBLE**

Target enclave mode is incompatible with the mode of the current RTS.  
Reason examples: 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**

Request to the AE service timed out.

#### **SGX\_ERROR\_SERVICE\_INVALID\_PRIVILEGE**

Request requires some special attributes for the enclave, but it does not privilege.

#### **SGX\_ERROR\_NDEBUG\_ENCLAVE**

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

#### **SGX\_ERROR\_UNDEFINED\_SYMBOL**

Enclave contains an import table.

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

#### **SGX\_ERROR\_INVALID\_MISC**

The MiscSelct or MiscMask settings are not correct.

#### **SGX\_ERROR\_HYPERV\_ENABLED**

Incompatible versions of Windows\* 10 OS and Hyper-V\* are detected. Disable Hyper-V on the target system.

#### **SGX\_ERROR\_PCL\_ENCRYPTED**

Enclave is encrypted but input parameters do not include the required content (e.g. sealed decryption key blob).

#### **SGX\_ERROR\_PCL\_NOT\_ENCRYPTED**

Enclave is not encrypted but input parameters include content for enclave decryption.

#### **SGX\_ERROR\_FEATURE\_NOT\_SUPPORTED**

Desired feature is not supported.

## SGX\_ERROR\_UNEXPECTED

Unexpected error is detected.

### Description

The `sgx_create_enclave_ex` function loads and initializes the enclave as described by `sgx_create_enclave`. In addition, `sgx_create_enclave_ex` activates extended features, based on the input provided in `ex_features` and `ex_features_p` parameters.

The following extended features are currently supported:

- Intel® SGX Protected Code Loader that enables loading encrypted enclaves.
- Switchless Calls. For more information, see the Switchless Calls section.
- Key Separation & Sharing (KSS). You can specify a different [sgx\\_kss\\_config\\_t](#) structure to load a KSS enabled enclave to have additional control options over the key derivation process. The KSS enabled enclave should be signed with `EnableKSS` set to 1 in the configuration file.

The described extended features are independent but can also work together.

### Requirements

Header	<code>sgx_urts.h</code>
Library	<code>sgx_urts.lib</code> or <code>sgx_urts_sim.lib</code> (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 [Loading and Unloading an Enclave](#).

### Requirements

Header	<code>sgx_urts.h</code>
Library	<code>sgx_urts.lib</code> or <code>sgx_urts_sim.lib</code> (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	<code>sgx_urts.h</code>
Library	<code>sgx_urts.lib</code>

**sgx\_select\_att\_key\_id**

`sgx_select_att_key_id` used to select the attestation key.

**Syntax**

```
sgx_status_t sgx_select_att_key_id(
    const uint8_t *p_att_key_id_list,
    uint32_t att_key_idlist_size,
    sgx_att_key_id_t *p_att_key_id
);
```

**Parameters****p\_att\_key\_id\_list [in]**

List of the supported attestation key IDs provided by the quote verifier.

### **att\_key\_id\_list\_size**

The size of `p_att_key_id_list`.

### **p\_att\_key\_id[out]**

Pointer to the selected attestation key. Cannot be NULL.

### **Return value**

#### **SGX\_SUCCESS**

All outputs are generated successfully.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

The `p_att_key_id_list` is not correct, list header is incorrect, the number of key IDs in the list exceeds the maximum or `p_att_key_id` pointer is invalid.

#### **SGX\_ERROR\_UNSUPPORTED\_ATT\_KEY\_ID**

The platform quoting infrastructure does not support the key described.

#### **SGX\_ERROR\_UNEXPECTED**

An unexpected error was detected.

### **Description**

You can select the attestation key id from the list provided by the off-platform Quote verifier. Calling `sgx_select_att_key_id` is the first thing an Intel® Software Guard Extensions application does when getting a quote of an enclave. Then call `sgx_init_quote_ext` to generate or obtain the attestation key. Calculate quote size by calling `sgx_get_quote_size_ex`. At last, call `sgx_get_quote_ext` to generate the quote.

The function will return a `sgx_att_key_id_t` of attestation keys supported both by the platform and the relying party. If the platform cannot support one in the list, the API will return error `SGX_ERROR_UNSUPPORTED_ATT_KEY_ID`. If the caller doesn't supply a list (`p_att_key_id_list == NULL`), then the platform software deem the relying party support all kinds of attestation keys. If there are multiple attestation keys are supported by both the platform and the relying party, in such case, if the "default quoting type" in registry(HKEY\_LOCAL\_MACHINE\SOFTWARE\Intel\SGX\_PSW\AESMQuotingType) is one of them, then the "default quoting type" will be returned; otherwise, the platform software will choose one of them according to its internal logic.

### **Requirements**

Header	sgx_uae_quote_ex.h
Library	sgx_quote_ex.lib or sgx_quote_ex_sim.lib (simulation)

**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 Intel® 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 Intel® 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\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

#### **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	<code>sgx_uae_epid.h</code>
Library	<code>sgx_epid.lib</code> or <code>sgx_epid_sim.lib</code> (simulation)

#### **sgx\_init\_quote\_ex**

Returns information required by an Intel® SGX application to get a quote of one of its enclaves.

#### Syntax

```
sgx_status_t sgx_init_quote_ex(
```

```

    const sgx_att_key_id_t *p_att_key_id,
    sgx_target_info_t *p_target_info,
    size_t* p_pub_key_id_size,
    uint8_t* p_pub_key_id
);

```

## Parameters

### **p\_att\_key\_id[in]**

Selected attestation key ID returned by `sgx_select_att_key_id`. Cannot be NULL.

### **p\_target\_info [out]**

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

### **p\_pub\_key\_id\_size [out]**

This parameter can be used in two ways. If `p_pub_key_id` is NULL, the API returns the buffer size required to hold the attestation public key ID. If `p_pub_key_id` is not NULL, `p_pub_key_size` must be large enough to hold the return attestation public key ID. Must not be NULL.

### **p\_pub\_key\_id [out]**

This parameter can be used in two ways. If it is passed in as NULL and `p_pub_key_id_size` is not NULL, the API returns the buffer size required to hold the attestation public key ID. If the parameter is not NULL, it must point to the buffer that is at least as long as the value passed in by `p_pub_key_id`.

## Return value

### **SGX\_SUCCESS**

All of the outputs are generated successfully.

### **SGX\_ERROR\_INVALID\_PARAMETER**

If `p_pub_key_id_size`, `p_att_key_id` is NULL, any of the other pointers are invalid. If `p_pub_key_size` is not NULL, any of the other pointers are invalid.

### **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**

Request to the AE service timed out.

**SGX\_ERROR\_OUT\_OF\_EPC**

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

**SGX\_ERROR\_UPDATE\_NEEDED**

Intel® SGX needs to be updated.

**SGX\_ERROR\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

**SGX\_ERROR\_UNSUPPORTED\_ATT\_KEY\_ID**

The platform quoting infrastructure does not support the key described.

**SGX\_ERROR\_ATT\_KEY\_CERTIFICATION\_FAILURE**

Failed to generate and certify the attestation key.

**SGX\_ERROR\_UNEXPECTED**

Unexpected error was detected.

**Description**

The application calls this API to request the owner of the selected platform attestation key to generate or obtain the attestation key.

If **SGX\_ERROR\_BUSY** is returned, you should wait (typically, several seconds to tens of seconds) and retry this API.

**Requirements**

Header	<code>sgx_uae_quote_ex.h</code>
Library	<code>sgx_quote_ex.lib</code> or <code>sgx_quote_ex_sim.lib</code> (simulation)

**sgx\_calc\_quote\_size**

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

**Syntax**

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

**Parameters****p\_sig\_rl [in]**

Optional revoke list of signatures, can be NULL.

**sig\_rl\_size [in]**

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

**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_calc_quote`, call `sgx_calc_quote_size` first to calculate the buffer size and then allocate enough memory for the quote.

**Requirements**

Header	<code>sgx_uae_epid.h</code>
Library	<code>sgx_epid.lib</code> or <code>sgx_epid_sim.lib</code> (simulation)

**sgx\_get\_quote\_size**

`sgx_get_quote_size` is deprecated. Use the `sgx_calc_quote_size` function instead.

`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	<code>sgx_uae_epid.h</code>
Library	<code>sgx_epid.lib</code> or <code>sgx_epid_sim.lib</code> (simulation)

**sgx\_get\_quote\_size\_ex**

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

**Syntax**

```
sgx_status_t sgx_get_quote_size_ex(
    const sgx_att_key_id_t *p_att_key_id,
    uint32_t *p_quote_size
);
```

**Parameters****p\_att\_key\_id[in]**

Selected attestation key ID returned by `sgx_select_att_key_id`. Cannot be NULL.

**p\_quote\_size [out]**

Indicate the size of quote buffer. Cannot be NULL.

**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.

**SGX\_ERROR\_ATT\_KEY\_UNINITIALIZED**

The platform quoting infrastructure does not have the attestation key available to generate quotes. Call `sgx_init_quote_ex` again.

**SGX\_ERROR\_UNSUPPORTED\_ATT\_KEY\_ID**

The platform quoting infrastructure does not support the key described.

**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_ex`, call `sgx_get_quote_size_ex` first to get the buffer size and then allocate enough memory for the quote.

**Requirements**

Header	sgx_uae_quote_ex.h
Library	sgx_quote_ex.lib or sgx_quote_ex_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_calc_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 Intel® EPID blob is corrupted.

**SGX\_ERROR\_EPID\_MEMBER\_REVOKED**

The Intel® 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\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

### **SGX\_ERROR\_UNEXPECTED**

An unexpected error was detected.

#### Description

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

The purpose of `p_qe_report` is for the ISV enclave to confirm the QUOTE it received is not modified by the untrusted SW stack, and not a replay. The implementation in QE is to generate a REPORT targeting the ISV enclave (target info from `p_report`), with the lower 32Bytes in `report.data = SHA256(p_nonce || p_quote)`. The ISV enclave can verify the `p_qe_report` and `report.data` to confirm the QUOTE has not be modified and is not a replay. It is optional.

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	<code>sgx_uae_epid.h</code>
Library	<code>sgx_epid.lib</code> or <code>sgx_epid_sim.lib</code> (simulation)

**sgx\_get\_quote\_ex**

`sgx_get_quote_ex` takes the application enclave REPORT and generates a QUOTE.

**Syntax**

```
sgx_status_t sgx_get_quote_ex(
    const sgx_report_t *p_app_report,
    const sgx_att_key_id_t *p_att_key_id,
    sgx_qe_report_info_t *p_qe_report_info,
    sgx_quote_t *p_quote,
    uint32_t quote_size
);
```

**Parameters****p\_app\_report [in]**

Report of the enclave for that the quote is being calculated. Cannot be NULL.

**p\_att\_key\_id[in]**

Selected attestation key ID returned by `sgx_select_att_key_id`. Cannot be NULL.

**p\_qe\_report\_info [in, out]**

Optional input and output contain the information required to generate a REPORT that can be verified by the application enclave.

**p\_quote [out]**

The major output of `sgx_get_quote_ex`, the quote itself cannot be NULL.

**quote\_size [in]**

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

**Return value****SGX\_SUCCESS**

All the outputs are generated successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

Any of the pointers are invalid.

**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**

Request to AE service timed out.

#### **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\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

#### **SGX\_ERROR\_UNSUPPORTED\_ATT\_KEY\_ID**

The platform quoting infrastructure does not support the key described.

#### **SGX\_ERROR\_ATT\_KEY\_UNINITIALIZED**

The platform quoting infrastructure does not have the attestation key available to generate quotes. Call `sgx_init_quote_ex` again.

#### **SGX\_ERROR\_UNEXPECTED**

Unexpected error was detected.

#### **Description**

The function takes the application enclave REPORT that will be converted into a quote after the QE verifies the REPORT. After the verification, the QE signs the REPORT with the platform attestation key matching the selected attestation key ID. If the key is not available, this API may return an error `SGX_ATT_KEY_NOT_INITIALIZED` depending on the algorithm. In this case, call `sgx_init_quote_ex` to re-generate and certify the attestation key.

The purpose of `qe_report` in `p_qe_report_info` is for the ISV enclave to confirm the QUOTE it received is not modified by the untrusted SW stack, and

not a replay. The implementation in QE is to generate a REPORT targeting the ISV enclave (**app\_enclave\_target\_info** from `p_app_report`), with the lower 32Bytes in `report.data = SHA256(nonce || p_quote)` (nonce from `p_app_report`). The ISV enclave can verify the `qe_report` and `report.data` to confirm the QUOTE has not been modified and is not a replay. It is optional.

If `SGX_ERROR_BUSY` is returned, you should wait (typically, several seconds to tens of seconds) and retry this API.

### Requirements

Header	<code>sgx_uae_quote_ex.h</code>
Library	<code>sgx_quote_ex.lib</code> or <code>sgx_quote_ex_sim.lib</code> (simulation)

### **sgx\_get\_supported\_att\_key\_id\_num**

`sgx_get_supported_att_key_id_num` returns the number of supported attestation key IDs on the platform.

### Syntax

```
sgx_status_t sgx_get_supported_att_key_id_num(
    uint32_t *p_att_key_id_num
);
```

### Parameters

#### **p\_att\_key\_id\_num[out]**

Indicate the pointer to the location where the required number will be returned.

### Return value

#### **SGX\_SUCCESS**

All the outputs are generated successfully.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

The `p_att_key_id_num` pointer is invalid.

#### **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\_UNEXPECTED**

An unexpected error was detected.

### Description

You need to call this function before getting supported attestation key IDs. The number is variable depending on the platform. You can use the returned `p_att_key_id_num` to allocate the buffer whose size is `sizeof sgx_att_key_id_ext_t * att_key_id_num` to hold the supported attestation key IDs.

### Requirements

Header	<code>sgx_uae_quote_ex.h</code>
Library	<code>sgx_quote_ex.lib</code> or <code>sgx_quote_ex_sim.lib</code> (simulation)

### **sgx\_get\_supported\_att\_key\_ids**

`sgx_get_supported_att_key_ids` returns an array of all supported attestation key IDs.

### Syntax

```
sgx_status_t sgx_get_supported_att_key_ids(
    sgx_att_key_id_ext_t *p_att_key_id_list,
    uint32_t att_key_id_num
);
```

### Parameters

#### **p\_att\_key\_id\_list [out]**

Pointer to the buffer that will contain supported attestation key IDs. The buffer size must be `sizeof sgx_att_key_id_ext_t * att_key_id_num`.

#### **att\_key\_id\_num [in]**

Indicate the the number of supported attestation key IDs. To get the number, call `sgx_get_supported_att_key_id_num` first.

### Return value

#### **SGX\_SUCCESS**

All the outputs are generated successfully.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

The `p_att_key_id_list` pointer is invalid or `att_key_id_num` is not correct.

#### **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\_UNEXPECTED**

An unexpected error was detected.

### Description

You can get all supported attestation key IDs on the platform used by `aesm_service`.

### Requirements

Header	<code>sgx_uae_quote_ex.h</code>
Library	<code>sgx_quote_ex.lib</code> or <code>sgx_quote_ex_sim.lib</code> (simulation)

#### **sgx\_ra\_get\_msg1**

`sgx_ra_get_msg1` is used to get the Intel® EPID 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 Intel® EPID blob is corrupted.

#### **SGX\_ERROR\_EPID\_MEMBER\_REVOKED**

The Intel® 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\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

#### **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	<code>sgx_ukey_exchange.h</code>
Library	<code>sgx_ukey_exchange.lib</code> or <code>sgx_ukey_exchangemt.lib</code>

### **sgx\_ra\_get\_msg1\_ex**

`sgx_ra_get_msg1_ex` is used to get the Intel® EPID or ECDSA remote attestation and key exchange protocol message 1 to send to a service provider. The application enclave should use `sgx_ra_init_ex` 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_ex(
    const sgx_att_key_id_t *p_att_key_id,
    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

#### **p\_att\_key\_id [in]**

Selected attestation key ID returned by `sgx_select_att_key_id`.

#### **context [in]**

Context returned by the `sgx_ra_init_ex` 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 Intel® EPID blob is corrupted.

### **SGX\_ERROR\_EPID\_MEMBER\_REVOKED**

The Intel® 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**

Request to AE service timed out.

### **SGX\_ERROR\_INVALID\_STATE**

The API is invoked in an incorrect order or state.

### **SGX\_ERROR\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

### **SGX\_ERROR\_UNSUPPORTED\_ATT\_KEY\_ID**

The platform quoting infrastructure does not support the key described.

### **SGX\_ERROR\_ATT\_KEY\_CERTIFICATION\_FAILURE**

Failed to generate and certify the attestation key.

### **SGX\_ERROR\_UNEXPECTED**

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.

If **SGX\_ERROR\_BUSY** is returned, you should wait (typically, several seconds to tens of seconds) and retry this API.

#### Requirements

Header	<code>sgx_ukey_exchange.h</code>
Library	<code>sgx_ukey_exchange.lib</code> or <code>sgx_ukey_exchangemt.lib</code>

### **sgx\_ra\_proc\_msg2**

`sgx_ra_proc_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_proc_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 Intel® EPID blob is corrupted.

**SGX\_ERROR\_EPID\_MEMBER\_REVOKED**

The Intel® 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\_KDF\_MISMATCH**

Indicates key derivation function does not match.

### **SGX\_ERROR\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

### **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	<code>sgx_ukey_exchange.h</code>
Library	<code>sgx_ukey_exchange.lib</code> or <code>sgx_ukey_exchangemt.lib</code>

### **sgx\_ra\_proc\_msg2\_ex**

`sgx_ra_proc_msg2_ex` 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_ex(
    const sgx_att_key_id_t *p_att_key_id,
    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

### **p\_att\_key\_id[in]**

Selected attestation key ID returned from `sgx_select_att_key_id`.

### **context [in]**

Context returned by `sgx_ra_init`.

### **eid [in]**

ID of the application enclave that 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_proc_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 Intel® EPID blob is corrupted.

#### **SGX\_ERROR\_EPID\_MEMBER\_REVOKED**

The Intel® 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**

Request to AE service timed out.

#### **SGX\_ERROR\_INVALID\_STATE**

The API is invoked in an incorrect order or state. Before calling this API, user should call `sgx_ra_get_msg1_ex` first.

#### **SGX\_ERROR\_INVALID\_SIGNATURE**

The signature is invalid.

**SGX\_ERROR\_MAC\_MISMATCH**

Indicates verification error for reports, sealed data, etc.

**SGX\_ERROR\_KDF\_MISMATCH**

Indicates key derivation function does not match.

**SGX\_ERROR\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

**SGX\_ERROR\_UNSUPPORTED\_ATT\_KEY\_ID**

The platform quoting infrastructure does not support the key described.

**SGX\_ERROR\_INVALID\_ATT\_KEY\_CERT\_DATA**

The data returned by the platform library's `sgx_get_quote_config` is invalid.

**SGX\_ERROR\_UNEXPECTED**

Unexpected error was detected.

**Description**

The `sgx_ra_proc_msg2_ex` processes the incoming message 2 and returns message 3. Message 3 is allocated by the library, so the caller should free it after use.

If **SGX\_ERROR\_BUSY** is returned, you should wait (typically, several seconds to tens of seconds) and retry this API.

**Requirements**

Header	<code>sgx_ukey_exchange.h</code>
Library	<code>sgx_ukey_exchange.lib</code> or <code>sgx_ukey_exchangemt.lib</code>

**sgx\_report\_attestation\_status**

`sgx_report_attestation_status` reports information from the Intel Attestation Server during a remote attestation to help to decide whether a TCB update is required. It is recommended to always call `sgx_report_attestation_status` after a remote attestation transaction when it results in a Platform Info Blob (PIB).

The `attestation_status` indicates whether the ISV server decided to trust the enclave or not.

- The value `pass:0` indicates that the ISV server trusts the enclave. If the ISV server trusts the enclave and platform services, `sgx_report_attestation_status` will not take actions to correct the TCB that will cause negative user experience such as long latencies or requesting a TCB update.
- The value `fail:!=0` indicates that the ISV server does not trust the enclave. If the ISV server does not trust the enclave or platform services, `sgx_report_attestation_status` will take all actions to correct the TCB which may incur long latencies and/or request the application to update one of the Intel SGX's TCB components. It is the ISV's responsibility to provide the TCB component updates to the client platform.

### 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 Intel® EPID blob is corrupted.

**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\_BUSY**

This service is temporarily unavailable.

**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\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

**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	<code>sgx_uae_service.h</code>
Library	<code>sgx_uae_service.lib</code> or <code>sgx_uae_service_sim.lib</code> (simulation)

**sgx\_check\_update\_status**

`sgx_check_update_status` reports information from the Intel Attestation Server during a remote attestation to help to learn whether a TCB update is available, and whether Intel® EPID provisioning or PSE provisioning/long-term pairing is or was needed or pending. It is recommended to always call `sgx_check_update_status` after a remote attestation transaction when it results in a Platform Info Blob (PIB).

**Syntax**

```
sgx_status_t sgx_check_update_status (
    const sgx_platform_info_t* p_platform_info,
    sgx_update_info_bit_t* p_update_info,
    uint32_t config,
    uint32_t* p_status
);
```

**Parameters****p\_platform\_info [in]**

Pointer to opaque structure received from Intel Attestation Server. Can be NULL when TCB is up to date. If it is, then `p_update_info` also needs to be NULL.

**p\_update\_info [out]**

Pointer to the buffer that receives the update information only when the return value of `sgx_check_update_status` is `SGX_ERROR_UPDATE_NEEDED`. Can be NULL.

**config [in]**

The value indicates whether caller wants to address pending Intel® EPID or PSE provisioning using the combination of the following bits.

Value	Description
bit 0:	reserved and must be zero.
bit 1:	set if caller wants to trigger Intel® EPID provisioning if it is needed/pending.
bit 2:	set if caller wants to trigger PSE provisioning/long-term pairing if it is needed/pending.
bit 31..3:	reserved and must be zero.
if bit[2:1] == 0:	never trigger either Intel® EPID or PSE provisioning/long-term pairing.

**p\_status [out]**

The value will be filled as follows:

Value	Description
bit 0:	set if any update is available. Caller can inspect <code>p_update_info</code> to learn details.
bit 1:	set if Intel® EPID provisioning is or was needed/pending. Set or cleared independent of config input.
bit 2:	set if PSE provisioning/long-term pairing is or was needed/pending. Set or cleared independent of config input.
bit 31..3:	reserved and must be zero.

Can be NULL. If all user wants is to know about updates, the API will return `SGX_ERROR_SERVICE_UNAVAILABLE` and fill in `p_update_info` even if `p_status` is NULL.

**Return value****SGX\_SUCCESS**

All the outputs are generated successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

Any of the pointers are invalid.

For example: `p_platform_info` NULL and `p_update_info` non-NULL (can't determine update status w/o PIB). Or `p_platform_info` NULL and `config == 0` (nothing to do).

**SGX\_ERROR\_UNSUPPORTED\_CONFIG**

Any unsupported bits in config input are set.

**SGX\_ERROR\_AE\_INVALID\_EPIDBLOB**

The Intel® EPID blob is corrupted.

**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\_BUSY**

This service is temporarily unavailable.

**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\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

**SGX\_ERROR\_UNEXPECTED**

An unexpected error was detected.

**Description**

The application calls `sgx_check_update_status` after remote attestation to help to recover the TCB and learn whether Intel® EPID provisioning or PSE provisioning/long-term pairing is or was needed/pending.

**Requirements**

Header	<code>sgx_uae_service.h</code>
Library	<code>sgx_uae_service.lib</code> or <code>sgx_uae_service_sim.lib</code> (simulation)

**sgx\_get\_extended\_epid\_group\_id**

The function `sgx_get_extended_epid_group_id` reports which extended Intel® EPID Group the client uses by default. The key used to sign a Quote will be a member of the extended Intel® EPID Group reported in this API.

**Syntax**

```
sgx_status_t sgx_get_extended_epid_group_id(
    uint32_t *p_extended_epid_group_id
);
```

## Parameters

### **p\_extended\_epid\_group\_id [out]**

The extended Intel® EPID Group ID.

## Return value

### **SGX\_SUCCESS**

All the outputs are generated successfully.

### **SGX\_ERROR\_INVALID\_PARAMETER**

The `p_extended_epid_group_id` pointer is invalid.

### **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\_UNEXPECTED**

An unexpected error was detected.

## Description

The application uses this value to tell the ISV Service Provider which extended Intel® EPID Group to use during remote attestation.

## Requirements

Header	<code>sgx_uae_service.h</code>
Library	<code>sgx_uae_service.lib</code> or <code>sgx_uae_service_sim.lib</code> (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 `sgx_ps_cap_t` 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	<code>sgx_uae_service.h</code>
Library	<code>sgx_uae_service.lib</code> or <code>sgx_uae_service_sim.lib</code> (simulation)

### **sgx\_register\_wl\_cert\_chain**

`sgx_register_wl_cert_chain` helps you to provide an Enclave Signing Key Allow List Certificate Chain. An Enclave Signing Key Allow 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 Allow 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 Allow List Cert Chain file.

#### **Return value**

##### **SGX\_SUCCESS**

All the outputs are generated successfully.

##### **SGX\_ERROR\_INVALID\_PARAMETER**

The Allow List is invalid.

##### **SGX\_ERROR\_SERVICE\_UNAVAILABLE**

The AE service did not respond.

##### **SGX\_ERROR\_SERVICE\_TIMEOUT**

The request to the AE service timed out.

##### **SGX\_ERROR\_UNEXPECTED**

An unexpected error is detected.

#### **Description**

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

#### **Requirements**

Header	sgx_uae_launch.h
Library	sgx_launch.dll

**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**

The platform is enabled for Intel SGX.

**SGX\_DISABLED\_REBOOT\_REQUIRED**

This platform is disabled for Intel SGX. It is configured to be enabled after the next reboot. The Intel SGX device is currently disabled and a reboot is required to enable it.

**SGX\_DISABLED\_MANUAL\_ENABLE**

The platform is disabled for Intel SGX but can be enabled manually through the BIOS menu. The Software Control Interface is not available to enable Intel SGX on this platform.

**SGX\_DISABLED\_LEGACY\_OS**

The operating system does not support UEFI enabling of the Intel SGX device. If UEFI is supported by the operating system in general, but support for enabling the Intel SGX device does not exist, this function will return the more general `SGX_DISABLED`.

**SGX\_DISABLED**

This platform is disabled for Intel SGX. More details about the ability to enable Intel® SGX are unavailable. There may be cases where Intel SGX can be manually enabled in the BIOS.

### 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\_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 Intel® SGX, and execution of this API will re-enable Intel® SGX but only after a reboot. If Intel® 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	<code>sgx_uae_service.h</code>
Library	<code>sgx_uae_service.dll</code>

---

**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 to check if the client platform is enabled for the Intel® Software Guard Extensions (Intel® SGX). You must run the client application with the administrator privileges to get the status successfully.

**Syntax**

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

**Parameters**

**sgx\_capable [out]**

Capable status of the Intel SGX device.

**1**

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

**0**

Intel SGX device is not available or may require manual configuration.

**Return value**

**SGX\_SUCCESS**

All the outputs are generated successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

`sgx_capable` pointer is invalid.

**SGX\_ERROR\_NO\_PRIVILEGE**

Application does not have the required privilege to read EFI variables. Run the application with administrator privileges to query the Intel SGX device status.

**SGX\_ERROR\_UNEXPECTED**

Unexpected error is detected.

## Description

ISV applications can call `sgx_is_capable` to detect if the Intel® SGX device is available. This API is intended to detect cases where software can configure the Intel SGX device. If 0 is returned, `sgx_cap_get_status` can be used to detect manual configuration changes that can be made to enable the Intel SGX device.

## Requirements

Header	<code>sgx_capable.h</code>
Library	<code>sgx_capable.dll</code>

### **NOTE:**

Administrative privileges are required to use this API.

### **sgx\_cap\_enable\_device**

`sgx_cap_enable_device` helps ISV applications to enable the Intel® Software Guard Extensions (Intel® SGX) device and return the appropriate status. If a reboot is required, an ISV application 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]**

Intel® SGX status of the Intel® SGX device.

#### **SGX\_ENABLED**

Platform is enabled for the Intel® SGX.

#### **SGX\_DISABLED\_REBOOT\_REQUIRED**

Platform is disabled for the Intel® SGX. Reboot required to enable the platform.

#### **SGX\_DISABLED\_MANUAL\_ENABLE**

Platform is disabled for the Intel® SGX but can be enabled manually through the BIOS menu. The Software Control Interface is not available to enable the Intel® SGX on this platform.

#### **SGX\_DISABLED\_HYPERV\_ENABLED**

Detected version of Windows\* OS10 is incompatible with the Hyper-V\*. The Intel® SGX cannot be enabled on the target system unless the Hyper-V\* is disabled.

#### **SGX\_DISABLED\_LEGACY\_OS**

Operating system does not support UEFI enabling of the the Intel® SGX device. If the operating system supports UEFI in general, but support for enabling the Intel® SGX device does not exist, this function returns `SGX_DISABLED`.

#### **SGX\_DISABLED\_UNSUPPORTED\_CPU**

Processor does not support the Intel SGX.

#### **SGX\_DISABLED**

Platform is disabled for the Intel® SGX. More details about enabling the Intel® SGX are unavailable. The Intel® SGX can be manually enabled in the BIOS.

#### [Return value](#)

#### **SGX\_SUCCESS**

All the outputs are generated successfully.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

The `sgx_device_status` pointer is invalid.

#### **SGX\_ERROR\_NO\_PRIVILEGE**

Application does not have the required privileges to read an UEFI variable. Run the application with the administrator privileges to enable the Intel® SGX device status.

#### **SGX\_ERROR\_HYPERV\_ENABLED**

Detected version of Windows\* OS 10 is incompatible with the Hyper-V\*. Disable the Hyper-V\* on the target system.

#### **SGX\_ERROR\_UNEXPECTED**

Unexpected error is detected.

#### [Description](#)

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

### Requirements

Header	<code>sgx_capable.h</code>
Library	<code>sgx_capable.dll</code>

---

#### **NOTE:**

Administrative privileges are required to use this API.

APIs that begin with `sgx_cap` are utility functions that operate independently of the Intel® SGX PSW. They do not require the PSW to be installed on the system. When the PSW is installed, they have the same behavior.

This API should be used inside APP installer to enable Intel® SGX device during installation.

---

#### **sgx\_cap\_get\_status**

`sgx_cap_get_status` helps ISV applications check the status of the Intel® Software Guard Extensions (Intel® SGX) on the client platform. You must run the client application with the administrator privileges to get the status successfully.

#### Syntax

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

#### Parameters

##### **sgx\_device\_status [out]**

Intel® SGX status of the Intel® SGX device.

##### **SGX\_ENABLED**

Platform is enabled for the Intel® SGX.

##### **SGX\_DISABLED\_REBOOT\_REQUIRED**

Platform is disabled for the Intel® SGX. Reboot required for enabling the platform.

##### **SGX\_DISABLED\_SCI\_AVAILABLE**

Platform is disabled for the Intel® SGX but can be enabled using the Software Control Interface.

#### **SGX\_DISABLED\_MANUAL\_ENABLE**

Platform is disabled for the Intel® SGX but can be enabled manually through the BIOS menu. The Software Control Interface is not available to enable the Intel® SGX on this platform.

#### **SGX\_DISABLED\_HYPERV\_ENABLED**

Detected version of Windows\* 10 is incompatible with Hyper-V\*. The Intel® SGX cannot be enabled on the target machine, unless Hyper-V\* is disabled.

#### **SGX\_DISABLED\_LEGACY\_OS**

Operating system does not support UEFI enabling of the Intel SGX device. If the operating system supports the UEFI in general but cannot enable the Intel® SGX device, the function returns `SGX_DISABLED`.

#### **SGX\_DISABLED\_UNSUPPORTED\_CPU**

Processor does not support the Intel® SGX.

#### **SGX\_DISABLED**

Platform is disabled for the Intel® SGX. You can try to enable the Intel® SGX manually through the BIOS menu.

#### [Return value](#)

#### **SGX\_SUCCESS**

All the outputs are generated successfully.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

The `sgx_device_status` pointer is invalid.

#### **SGX\_ERROR\_NO\_PRIVILEGE**

Application does not have the required privileges to read EFI variables. Run the application with the administrator privileges to query the Intel® SGX device status.

#### **SGX\_ERROR\_UNEXPECTED**

Unexpected error is detected.

#### [Description](#)

ISV applications can call `sgx_cap_get_status` to detect if the Intel® SGX is enabled or can be enabled on the device, using the software interface or by taking manual configuration steps.

### Requirements

Header	<code>sgx_capable.h</code>
Library	<code>sgx_capable.dll</code>

---

#### **NOTE:**

Administrator privileges are required to use this API.

APIs that begin with `sgx_cap` are the utility functions that operate independently of the Intel® SGX Platform Software (Intel® SGX PSW). They do not require the Intel® SGX PSW to be installed on the system. When the Intel® SGX PSW is installed, the functions behavior remains unchanged.

This API should be used inside APP installer to query Intel® SGX device status during installation.

---

#### **sgx\_cap\_get\_psw\_version\_string**

`sgx_cap_get_psw_version_string` helps ISV applications determine the version of the Intel® SGX Platform Software (PSW) installed on the device. A buffer is specified along with the buffer length (in characters). If the buffer is not large enough to hold the PSW version string or is NULL, the buffer length needed to store the version string is returned in the size parameter.

#### Syntax

```
sgx_status_t sgx_cap_get_psw_version_stringw(
    LPWSTR *version_string,
    DWORD *version_string_len
);
```

or

```
sgx_status_t sgx_cap_get_psw_version_stringa(
    LPSTR *version_string,
    DWORD *version_string_len
);
```

#### Parameters

##### **version\_string [out]**

Version of the PSW installed on the system, in the string format.

**version\_string\_size [in out]**

When `version_string` is not NULL, `version_string_length` is interpreted as the length of `version_string`. If `version_string_len` is too small to hold the string, or if a NULL pointer is passed in for `version_string`, `version_string_len` returns the length required.

**Return value****SGX\_SUCCESS**

All the outputs are generated successfully. A NULL terminated version string is returned in the `version_string` parameter.

**SGX\_ERROR\_INVALID\_PARAMETER**

The `version_string_len` pointer is invalid.

**SGX\_ERROR\_OUT\_OF\_MEMORY**

There is no sufficient space in the `version_string` parameter. The size of the buffer needed is returned in the `version_string_len` parameter.

**SGX\_ERROR\_SERVICE\_UNAVAILABLE**

The Intel SGX PSW is not installed. As a result, the version string is not available.

**SGX\_ERROR\_UNEXPECTED**

An unexpected error is detected.

**Description**

ISV applications can call `sgx_cap_get_psw_version_string` to determine the version of the Intel SGX Platform Software installed on the device.

**Requirements**

Header	<code>sgx_capable.h</code>
Library	<code>sgx_capable.dll</code>

**NOTE:**

APIs that begin with `sgx_cap` are utility functions that operate independently of the Intel SGX PSW. They do not require the PSW to be installed on the system. When the PSW is installed, they have the same behavior.

### **sgx\_get\_whitelist\_size**

`sgx_get_whitelist_size` returns the required buffer size for the allowlist.

#### Syntax

```
sgx_status_t sgx_get_whitelist_size(  
    uint32_t *p_whitelist_size  
);
```

#### Parameters

##### **p\_whitelist\_size [out]**

Indicate the size of the allowlist buffer.

#### Return value

##### **SGX\_SUCCESS**

All the outputs are generated successfully.

##### **SGX\_ERROR\_INVALID\_PARAMETER**

The `p_whitelist_size` pointer is invalid.

##### **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\_UNEXPECTED**

An unexpected error was detected.

#### 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_whitel-`

ist, call `sgx_get_whitelist_size` first to get the buffer size and then allocate enough memory for the quote.

### Requirements

Header	<code>sgx_uae_launch.h</code>
Library	<code>sgx_launch.lib</code> or <code>sgx_launch_sim.lib</code> (simulation)

### **sgx\_get\_whitelist**

`sgx_get_whitelist` returns the allowlist used by `aesm_service`.

### Syntax

```
sgx_status_t sgx_get_whitelist(
    uint8_t *p_whitelist,
    uint32_t whitelist_size
);
```

### Parameters

#### **p\_whitelist [out]**

The allowlist.

#### **whitelist\_size [in]**

Indicate the size of the allowlist buffer. To get the size, call `sgx_get_whitelist_size` first.

### Return value

#### **SGX\_SUCCESS**

All the outputs are generated successfully.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

The `p_whitelist` pointer is invalid or `whitelist_size` is not correct.

#### **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\_UNEXPECTED**

An unexpected error was detected.

**Description**

You can get current allowlist used by `aesm_service`.

**Requirements**

Header	<code>sgx_uae_launch.h</code>
Library	<code>sgx_launch.lib</code> or <code>sgx_launch_sim.lib</code> (simulation)

**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 the calling 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 calling 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. `sgx_is_within_enclave` returns 0 if the buffer is outside the enclave or overlaps with the enclave boundary. Thus `!sgx_is_within_enclave() ≠ sgx_is_outside_enclave()`.

## Requirements

Header	<code>sgx_trts.h</code>
Library	<code>sgx_trts.lib</code> or <code>sgx_trts_sim.lib</code> (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 the calling 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 calling enclave address space. It does not check the property of the address. `sgx_is_outside_enclave` returns 0 if the buffer is inside the enclave or overlaps with the enclave boundary. Thus `!sgx_is_outside_enclave() ≠ sgx_is_within_enclave()`.

### Requirements

Header	<code>sgx_trts.h</code>
Library	<code>sgx_trts.lib</code> or <code>sgx_trts_sim.lib</code> (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. The rand buffer can be either within or outside the enclave, but it is not allowed to be across the enclave boundary or wrapped around.

#### **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	<code>sgx_trts.h</code>
Library	<code>sgx_trts.lib</code> or <code>sgx_trts_sim.lib</code> (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]**

Specify 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 SDK supports the registration of custom exception handler functions. 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 nonzero) or append the handler to the handler chain.

After calling `sgx_register_exception_handler` to prepend an exception handler, a subsequent call to this function may add another exception handler at the beginning of the handler chain. Therefore the order in which exception handlers are called does not only depend on the value of the `is_first_handler` parameter, but more importantly depends on the order in which exception handlers are registered.

---

### **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	<code>sgx_trts_exception.h</code>
Library	<code>sgx_trts.lib</code> or <code>sgx_trts_sim.lib</code> (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 the registration of custom exception handler functions. 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	<code>sgx_trts_exception.h</code>
Library	<code>sgx_trts.lib</code> or <code>sgx_trts_sim.lib</code> (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	<code>sgx_spinlock.h</code>
Library	<code>sgx_tstdc.lib</code>

### **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	<code>sgx_spinlock.h</code>
Library	<code>sgx_tstdc.lib</code>

## 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_mutex_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`, 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.

---

#### **NOTE**

---

---

A thread should not exit an enclave returning from a root ECALL after acquiring the ownership of a mutex. Do not split the critical section protected by a mutex across root ECALLs.

---

### 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.

---

### **NOTE**

A thread should not exit an enclave returning from a root ECALL after acquiring the ownership of a mutex. Do not split the critical section protected by a mutex across root ECALLs.

---

### 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	<code>sgx_thread.h</code> <code>sgxtstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

### `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	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

### **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	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

**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	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

### **sgx\_thread\_equal**

The `sgx_thread_equal` function compares two thread identifiers.

#### Syntax

```
int sgx_thread_equal(sgx_thread_t
    sgx_thread_t t1,
    sgx_thread_t t2
);
```

#### Return value

A nonzero value if the two thread IDs are equal, 0 otherwise.

#### Description

The function compares two thread identifiers provided by `sgx_thread_self` to determine if the IDs refer to the same trusted thread.

### Requirements

Header	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

### **sgx\_create\_event**

The `sgx_create_event` function initializes an unnamed event within the enclave.

#### Syntax

```
int sgx_create_event(
    sgx_thread_sync_object_handle_t* handle,
    int is_manual_reset,
    int is_signaled_init_state
```

```
) ;
```

## Parameters

### **handle [out]**

Pointer to handle of the trusted thread synchronization object (event).

### **is\_manual\_reset [in]**

If this parameter is 1, the event can only be reset manually.

If this parameter is 0, the event will be reset automatically.

### **is\_signaled\_init\_state [in]**

By default, the initial state of the event is not set. However, when this parameter is 1, the event will be signaled upon initialization.

When this parameter is 0, the event will be non-signaled upon initialization.

## Return value

### **0**

The unnamed event is initialized successfully.

### **EINVAL**

The pointer to the trusted event is invalid.

### **ENOMEM**

There is not enough memory to allocate an event.

## Description

When a thread creates an event within an enclave, `sgx_create_event` simply initializes various fields of the event object to indicate that the event is available. By default, `sgx_create_event` creates a non-signaled event that is automatically reset by `sgx_wait_for_single_object` and `sgx_wait_for_multiple_objects`. The results of using an event in any operation before it has been fully initialized are undefined.

## Requirements

Header	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

**sgx\_set\_event**

The `sgx_set_event` function sets the event object to the signaled state.

**Syntax**

```
int sgx_set_event(
    sgx_thread_sync_object_handle_t handle
);
```

**Parameters****handle [in]**

Handle of the trusted thread synchronization object (event).

**Return value**

**0**

The event object state was signaled successfully.

**EINVAL**

The pointer to the trusted event is invalid.

**Description**

`sgx_set_event` explicitly signals the state of an event object. The state of an auto-reset event object remains signaled until a thread waiting on this event is released. The state of a manually-reset event object remains signaled until you explicitly reset it calling `sgx_reset_event`. All threads waiting on a manually-reset event will be released.

**Requirements**

Header	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

**sgx\_reset\_event**

The `sgx_reset_event` function sets the event object to the non-signaled state.

**Syntax**

```
int sgx_reset_event(
    sgx_thread_sync_object_handle_t handle
);
```

## Parameters

### **handle [in]**

Handle of the trusted thread synchronization object (event).

## Return value

### **0**

The event object state is reset successfully.

### **EINVAL**

The pointer to the trusted event is invalid.

## Description

`sgx_reset_event` changes the state of an event object to non-signaled. Use this function to reset events configured not to be automatically reset.

## Requirements

Header	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

### **sgx\_wait\_for\_single\_object**

The `sgx_wait_for_single_object` function waits until the specified object is in the signaled state or the timeout interval elapses.

## Syntax

```
int sgx_wait_for_single_object(
    sgx_thread_sync_object_handle_t handle,
    uint32_t timeout
);
```

## Parameters

### **handle [in]**

Handle of the trusted thread synchronization object (event).

### **timeout [in]**

Timeout interval. Only 0 and `WAIT_FUNC_TIMEOUT_INFINITE` are supported. With `WAIT_FUNC_TIMEOUT_INFINITE` a thread waits until the

object is signaled. If timeout is zero, the function does not enter a wait state if the object is not signaled. Instead it returns immediately.

### Return value

#### 0

The object was signaled successfully.

#### **SGX\_THREAD\_ERROR\_WAIT\_TIMEOUT**

A timeout interval of 0 was specified and the object was not in the signaled state.

#### **SGX\_THREAD\_ERROR\_WAIT\_FAILED**

The function failed to wait.

### Description

`sgx_wait_for_single_object` allows a thread to wait until an object is signaled (timeout `WAIT_FUNC_TIMEOUT_INFINITE`) or to check whether an object has been signaled (timeout zero) without waiting if not.

### Requirements

Header	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

#### **sgx\_wait\_for\_multiple\_objects**

The `sgx_wait_for_multiple_objects` function waits until one or all of the specified objects are in the signaled state or the timeout interval elapses.

### Syntax

```
int sgx_wait_for_multiple_objects(
    uint32_t count,
    sgx_thread_sync_object_handle_t * handles,
    int wait_all,
    uint32_t timeout
);
```

### Parameters

#### **count [in]**

Number of objects in the handle array. The current implementation limits the number of objects to `MAX_WAIT_HANDLE_NUM`.

**handles [in]**

Array of handles of the trusted thread synchronization object (event).

**wait\_all [in]**

If this parameter is 1, `sgx_wait_for_multiple_objects` returns only after all the objects in the handle array are signaled. When this parameter is 0, the function returns when the state of any object is signaled. In this case, the return value indicates the object whose state caused the function to return.

**timeout [in]**

Timeout interval. Only 0 and `WAIT_FUNC_TIMEOUT_INFINITE` are supported. With `WAIT_FUNC_TIMEOUT_INFINITE`, a thread waits until the event object is signaled. If timeout is zero, the function does not enter a wait state if the event object is not signaled. Instead it returns immediately.

**Return value****0**

If the function succeeds.

**SGX\_THREAD\_ERROR\_WAIT\_TIMEOUT**

A timeout interval of 0 was specified and no object was in the signaled state.

**SGX\_THREAD\_ERROR\_WAIT\_FAILED**

The function failed to wait.

**Description**

`sgx_wait_for_multiple_objects` allows a thread to wait until an object is signaled (timeout `WAIT_FUNC_TIMEOUT_INFINITE`) or to check whether if an event object has been signaled (timeout zero). Handles could refer to different object types: event and mutex.

**Requirements**

Header	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

**sgx\_destroy\_event**

The `sgx_destroy_event` function destroys the event object and releases all its resources.

**Syntax**

```
int sgx_destroy_event(
    sgx_thread_sync_object_handle_t handle
);
```

## Parameters

### handle [in]

Handle of the trusted thread synchronization object (event).

## Return value

### 0

The event object was destroyed successfully.

### Non zero

The function failed. See the returned error code for additional information.

## Description

`sgx_destroy_event` destroys an event object and releases all the resources associated with the event. This function signals all threads waiting on the event before it is closed.

---

### **NOTE:**

The `sgx_destroy_event` returns different values if the event object to be destroyed is being waited by `sgx_wait_single_object` or `sgx_wait_multiple_objects`:

- The `sgx_destroy_event` returns `EBUSY` if the event is being waited by `sgx_wait_single_object`.
  - The `sgx_destroy_event` returns `0` if the event is being waited by `sgx_wait_multiple_objects`.
- 

## Requirements

Header	<code>sgx_thread.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

### **sgx\_cpuid**

The `sgx_cpuid` function performs the equivalent of a `cpuid()` function call or intrinsic 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 leaves. (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 intrinsic 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 leaves. (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	<code>sgx_cpuid.h</code> <code>sgx_tstdc.edl</code>
Library	<code>sgx_tstdc.lib</code>

**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 Intel 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` and `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.

---

#### **NOTE:**

It is not recommended to use this API to obtain the sealing key. Use the `sgx_seal_data`, `sgx_seal_data_ex`, and `sgx_unseal_data` API instead. The sealing key can change after the platform firmware is updated. The sealing data API generates a data blob (`sgx_sealed_data_t`), which contains all the necessary information to unseal the blob even after updating the platform firmware. Without this information, unsealing may fail.

---

#### Requirements

Header	<code>sgx_utils.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

#### **sgx\_create\_report**

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 Intel® Software Guard Extensions (Intel® 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]**

Pointer to the [sgx\\_target\\_info\\_t](#) object that contains the information of the target enclave, which will cryptographically verify the report by calling `sgx_verify_report`.

- If the pointer value is `NULL`, `sgx_create_report` retrieves information about the calling enclave, but the generated report cannot be verified by any enclave.
- If the pointer value is *not* `NULL`, the `target_info` buffer must be within the enclave.

See [sgx\\_target\\_info\\_t](#) for structure details.

### **report\_data [in]**

Pointer to the [sgx\\_report\\_data\\_t](#) object that 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]**

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` but the memory is not within the enclave or the reserved fields of the data structure are not set to zero.

## Description

Use the function `sgx_create_report` to create a cryptographic report that describes the contents of the calling enclave. The report can be used by other enclaves to verify that the enclave is running on the same platform. When an enclave calls `sgx_verify_report` to verify a report, it succeeds only if the

report has been generated using the `target_info` for said enclave. This function is a wrapper for the Intel® SGX `EREPORT` instruction.

Before the source enclave calls `sgx_create_report` to generate a report, it needs to populate `target_info` with information about the target enclave that will verify the report. The target enclave may obtain this information by calling `sgx_create_report` with a `NULL` pointer or directly calling `sgx_self_target` for `target_info` and pass it to the source enclave at the beginning of the inter-enclave attestation process.

## Requirements

Header	<code>sgx_utils.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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: The input report was generated using a `target_info` that matches the one for the enclave making this call.

### **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	<code>sgx_utils.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

### **sgx\_self\_report**

Generates a self cryptographic report of an enclave.

#### Syntax

```
const sgx_report_t *sgx_self_report(void);
```

#### Return value

The function returns a constant pointer to the generated self cryptographic report of an enclave. See `sgx_report_t` for structure details.

#### Description

This function returns a self cryptographic report of an enclave. On the first call, the function calls `sgx_create_report` with a `NULL` pointer for `target_info` to generate a self cryptographic report of the enclave and saves it. For the subsequent calls, the function directly returns the generated report pointer.

#### Requirements

Header	<code>sgx_utils.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

**sgx\_self\_target**

Generates self target info from the self cryptographic report of an enclave.

**Syntax**

```
sgx_status_t sgx_self_target(
    sgx_target_info_t *target_info
);
```

**Parameters****target\_info [OUT]**

Pointer to the [sgx\\_target\\_info\\_t](#) object that receives the generated self target info from the self report of an enclave. The `target_info` must be a non-NULL pointer, and the buffer must be located within the enclave.

**Return value****SGX\_SUCCESS**

Indicates success.

**SGX\_ERROR\_INVALID\_PARAMETER**

Invalid input parameters detected.

**SGX\_ERROR\_UNEXPECTED**

Indicates an unexpected error occurred.

**Description**

The function `sgx_self_target` generates self target info with the self cryptographic report of the enclave. You can use it to get target info in the inter-enclave attestation process.

**Requirements**

Header	<code>sgx_utils.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

**sgx\_get\_aligned\_ptr**

The `sgx_get_aligned_ptr` function returns the address within the provided buffer. The returned address will be used as the starting address of the structure to be aligned.

## Syntax

```
void *sgx_get_aligned_ptr (
    void *raw,
    size_t raw_size,
    size_t allocate_size,
    size_t alignment,
    align_req_t *data,
    size_t count
);
```

## Parameters

### **raw [in]**

Pointer to the buffer allocated by the caller.

### **raw\_size [in]**

Size of the raw buffer.

### **allocate\_size [in]**

Size of the structure to be aligned.

### **alignment [in]**

Desired, traditional alignment of the structure. Must be power of 2.

### **data [in]**

(offset, length) pairs to define the fields in the structure that needs confidentiality protection. If data is NULL and count is 0, the whole structure is treated as needing confidentiality protection.

### **count [in]**

Number of `align_req_t` structures in the data.

## Return value

Address within the provided buffer where structure to be aligned must start or NULL, including the case when the structure cannot be aligned.

## Description

The `sgx_get_aligned_ptr` function facilitates alignment of the structure that contains secrets. The function returns the address within the provided raw buffer to be used as the start address of the structure on a specific boundary. If the structure cannot be aligned, the function returns NULL.

If the whole structure cannot be aligned, you can use `align_req_t` structure to define part of the secrets in the structure to be protected.

The raw buffer is defined/allocated by the caller. In general, its size (specified by the `raw_size` parameter) must be bigger than the structure being aligned. The delta between the raw buffer size and the structure size depends on value of the desired, traditional alignment.  $\text{Raw\_size} \geq \text{sizeof}(\text{structure}) + 64 + A$ , where  $A = \max((\text{desired, traditional alignment}), 8)$

## Requirements

Header	<code>sgx_secure_align_api.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

## `sgx_aligned_malloc`

The `sgx_aligned_malloc` function allocates memory for a structure on a specified alignment boundary and returns the address where structure must start in order to be aligned.

## Syntax

```
void *sgx_aligned_malloc (
    size_t size,
    size_t alignment,
    align_req_t *data,
    size_t count
);
```

## Parameters

### **size [in]**

Size of the structure to be aligned.

### **alignment [in]**

Desired, traditional alignment of the structure. Must be power of 2.

### **data [in]**

(offset, length) pairs to define the fields in the structure that needs confidentiality protection. If the data is NULL and count is 0, the whole structure is treated as needing confidentiality protection.

### count [in]

Number of `align_req_t` structures in the data.

### Return value

Pointer to the memory block that is allocated or NULL if the operation failed, including the case when the structure cannot be aligned.

### Description

The `sgx_aligned_malloc` function allocates memory for the structure that contains secrets on a specified alignment boundary. If the structure cannot be aligned, the function returns NULL.

If the whole structure cannot be aligned, you can use the `align_req_t` structure to define part of the secrets in the structure to be protected.

The pointer allocated by `sgx_aligned_malloc` must be released by `sgx_aligned_free`.

### Requirements

Header	<code>sgx_secure_align_api.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

### sgx\_aligned\_free

The `sgx_aligned_free` function frees a block of memory allocated by `sgx_aligned_malloc`.

### Syntax

```
void *sgx_aligned_free (
    void *size,
    size_t alignment,
    align_req_t *data,
```

```

        size_t count
    );

```

## Parameters

### **ptr [in]**

Pointer to the memory block that has been returned to `sgx_aligned_malloc`.

## Description

The `sgx_aligned_free` function frees the memory allocated by `sgx_aligned_malloc`. It does not check the input parameter. If the input pointer has not been previously allocated by `sgx_aligned_malloc`, the result is unpredictable.

## Requirements

Header	<code>sgx_secure_align_api.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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 it 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	<code>sgx_tseal.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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 it 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	<code>sgx_tseal.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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. Otherwise, the return value is `0xFFFFFFFF`. It is recommended that you check the return value before use it 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	<code>sgx_tseal.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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\_additional\_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	<code>sgx_tseal.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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 is detailed in [sgx\\_key\\_request\\_t](#).

### **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=0xFF0000000000000B`, `xfrm=0`.

### **misc\_mask [in]**

Identifies the mask bits for the Misc feature to enforce. Function `sgx_seal_data` uses `0xF0000000`. 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\_additional\_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	<code>sgx_tseal.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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_additional_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_add_mac_txt_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_add_mac_txt_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 cross the enclave boundary.
- `p_decrypted_text` and `p_decrypted_text_length` must be within the enclave.
- `p_decrypted_text` and `p_additional_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_add_mac_txt_len` and `sgx_get_encrypt_txt_len` are provided. The decrypted text buffer must be allocated within the enclave.

#### **Requirements**

Header	<code>sgx_tseal.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

#### **sgx\_mac\_aadata**

This function is used to authenticate the input data with AES-GMAC.

## Syntax

```
sgx_status_t sgx_mac_aadata (
    const uint32_t additional_MACtext_length,
    const uint8_t * p_additional_MACtext,
    const uint32_t sealed_data_size,
    sgx_sealed_data_t * p_sealed_data
);
```

## Parameters

### **additional\_MACtext\_length [in]**

Length of the plain text to provide authentication for in bytes.

### **p\_additional\_MACtext [in]**

Pointer to the plain text to provide authentication for.

---

**NOTE:**

This data is not encrypted. This data can be within or outside the enclave, but cannot cross the enclave boundary.

---

### **sealed\_data\_size [in]**

Number of bytes allocated for the `sealed_data_t` structure. The calling code should utilize the 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_t` structure.

---

**NOTE:**

The calling code must allocate the memory for this buffer and should utilize the helper function `sgx_calc_sealed_data_size` with 0 as the `txt_encrypt_size` to determine the required buffer size. The `sealed_data_t` structure 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:

- `p_additional_mactext` buffer can be within or outside the enclave, but cannot cross the enclave boundary.
- `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_mac_aadata` function retrieves a key unique to the enclave and uses that key to generate the authentication tag based on the input data buffer. This function can be utilized to provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted. The data origin authentication can be demonstrated on future instantiations of the enclave using the MAC stored into the data blob.

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 must be allocated within the enclave.

#### Requirements

Header	<code>sgx_tseal.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

#### **sgx\_mac\_aadata\_ex**

This function is used to authenticate the input data with AES-GMAC. This is the expert mode version of the function `sgx_mac_aadata`.

#### Syntax

```
sgx_status_t sgx_mac_aadata_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 sealed_data_size,
    sgx_sealed_data_t * p_sealed_data
);

```

## Parameters

### **key\_policy [in]**

Specifies the policy to use in the key derivation. Key policy name Value Description is detailed in [sgx\\_key\\_request\\_t](#). Function `sgx_mac_aadata` uses the MRSIGNER policy.

### **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_mac_aadata` uses `flags=0xffffffffffff3, xfrm=0`.

### **misc\_mask [in]**

The `MISC_SELECT` mask bits for the enclave. Reserved for future function extension.

### **additional\_MACtext\_length [in]**

Length of the plain text data stream to be MAC'ed in bytes.

### **p\_addtional\_MACtext [in]**

Pointer to the plain text data stream to be MAC'ed of variable length.

---

### **NOTE:**

This data is not encrypted. This data can be within or outside the enclave, but cannot cross the enclave boundary.

---

### **sealed\_data\_size [in]**

Number of bytes allocated for the `sealed_data_t` structure. The calling code should utilize the 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 the helper function `sgx_calc_sealed_data_size` with 0 as the `txt_encrypt_size` to determine the required buffer size. The `sealed_data_t` structure 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:

- `p_additional_mactext` buffer can be within or outside the enclave, but cannot cross the enclave boundary.
- `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_mac_aadata_ex` is an extended version of `sgx_mac_aadata`. 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_mac_aadata` and the default values provided for `key_policy` (`MR_SIGNER`) and an attribute mask which includes the `RESERVED`, `INITED` and `DEBUG` bits. Before you use this function, you should have a clear understanding of the impact of using a policy and/or `attribute_mask` that is different from that in `sgx_mac_aadata`.

**Requirement**

Header	sgx_tseal.h
Library	sgx_tservice.lib or sgx_tservice_sim.lib (simulation)

**sgx\_unmac\_aadata**

This function is used to verify the authenticity of the input sealed data structure using AES-GMAC. This function verifies the MAC generated with `sgx_mac_aadata` or `sgx_mac_aadata_ex`.

**Syntax**

```
sgx_status_t sgx_unmac_aadata(
    const sgx_sealed_data_t * p_sealed_data,
    uint8_t * p_additional_MACtext,
    uint32_t * p_additional_MACtext_length,
);
```

**Parameters****p\_sealed\_data [in]**

Pointer to the sealed data structure to be authenticated with AES-GMAC. Must be within the enclave.

**p\_additional\_MACtext [out]**

Pointer to the plain text data stream that was AES-GMAC protected. You should call the helper function `sgx_get_add_mac_txt_len` to determine the required buffer size to allocate.

**p\_additional\_MACtext\_length [in, out]**

Pointer to the length of the plain text data stream in bytes. Upon successful tag matching, `sgx_unmac_aadata` sets this parameter with the actual length of the plaintext stored in `p_additional_MACtext`.

**Return value****SGX\_SUCCESS**

The authentication tag in the `sealed_data_t` structure matches the expected value.

**SGX\_ERROR\_INVALID\_PARAMETER**

This parameter indicates an error if the parameters do not meet any of the following conditions:

- `p_additional_MACtext` buffers can be within or outside the enclave, but cannot cross the enclave boundary.
- `p_addtitional_MACtext` buffers must be big enough to receive the plain text data.
- `p_sealed_data` buffers must be within the enclave.
- Input buffers cannot cross an enclave boundary.

### **SGX\_ERROR\_INVALID\_CPUSVN**

The CPUSVN in the data blob is beyond the CPUSVN value of the platform.

### **SGX\_ERROR\_INVALID\_ISVSVN**

The ISVSVN in the data blob is greater than the ISVSVN value of the enclave.

### **SGX\_ERROR\_MAC\_MISMATCH**

The tag verification fails. The error may be caused by a platform update, software update, or corruption of the `sealed_data_t` structure.

### **SGX\_ERROR\_OUT\_OF\_MEMORY**

The enclave is out of memory.

### **SGX\_ERROR\_UNEXPECTED**

Indicates a cryptography library failure.

### Description

The `sgx_unmac_aadata` function verifies the tag with AES-GMAC. Use this function to demonstrate the authenticity of data that was preserved by an earlier instantiation of this enclave.

You need to allocate additional data buffer. To determine the minimum memory to allocate for additional data buffers, use the helper function `sgx_get_add_mac_txt_len`.

### Requirements

Header	<code>sgx_tseal.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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, for 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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

### **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 step 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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**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_t sha_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

### 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_t sha_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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

## **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_t sha_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 datasets. The caller uses this function to deallocate memory used to store the SHA256 calculation state.

## Requirements

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

### **sgx\_sha384\_msg**

The `sgx_sha384_msg` function performs a standard SHA384 hash over the input data buffer.

## Syntax

```
sgx_status_t sgx_sha384_msg(
    const uint8_t *p_src,
    uint32_t src_len,
    sgx_sha384_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 384bit hash resulting from the SHA384 calculation. This pointer must be non-NULL and the caller allocates memory for this buffer.

## Return value

### **SGX\_SUCCESS**

The SHA384 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 SHA384 hash calculation failed.

**Description**

The `sgx_sha384_msg` function performs a standard SHA384 hash over the input data buffer. Only a 384-bit version of the SHA hash is supported. (Other sizes, for 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 SHA384 bit hash over multiple input data sets.

A zero-length input data buffer is supported but the pointer must be non-NULL.

**Requirements**

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**sgx\_sha384\_init**

`sgx_sha384_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_sha384_msg` to perform the hash in a single call.

**Syntax**

```
sgx_status_t sgx_sha384_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 SHA384 hash. The algorithm stores the intermediate results of performing the hash calculation over data sets.

## Return value

### **SGX\_SUCCESS**

The SHA384 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 SHA384 state is not initialized properly due to an internal cryptography library failure.

## Description

Calling `sgx_sha384_init` is the first step in performing a SHA384 hash over multiple datasets. The caller does not allocate memory for the SHA384 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_sha384_close` should be called to free the state allocated by this algorithm.

## Requirements

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**sgx\_sha384\_update**

`sgx_sha384_update` performs a SHA384 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_sha384_update (
    const uint8_t *p_src,
    uint32_t src_len,
    sgx_sha_state_handle_t sha_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 SHA384 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 SHA384 hash calculation.

## Description

This function should be used as part of a SHA384 calculation over multiple datasets. If a SHA384 hash is needed over a single data set, function `sgx_sha384_msg` should be used instead. Prior to calling this function on the first dataset, the `sgx_sha384_init` function must be called first to allocate and initialize the SHA384 state structure which will hold intermediate hash results over earlier datasets. The function `sgx_sha384_get_hash` should be used to obtain the hash after the final dataset has been processed by this function.

## Requirements

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

### **sgx\_sha384\_get\_hash**

`sgx_sha384_get_hash` obtains the SHA384 hash after the final dataset has been processed (by calls to `sgx_sha384_update`).

## Syntax

```
sgx_status_t sgx_sha384_get_hash(
    sgx_sha_state_handle_t sha_handle,
    sgx_sha384_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 SHA384 hash. The algorithm stores the intermediate results of performing the hash calculation over multiple datasets.

### **p\_hash [out]**

This is a pointer to the 384-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 SHA384 state passed in is likely problematic causing an internal cryptography library failure.

## Description

This function returns the hash after performing the SHA384 calculation over one or more datasets using the `sgx_sha384_update` function. Memory for the hash should be allocated by the calling function. The handle to SHA384 state used in the `sgx_sha384_update` calls must be passed in as input.

## Requirements

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

### **sgx\_sha384\_close**

`sgx_sha384_close` cleans up and deallocates the SHA384 state that was allocated in function `sgx_sha384_init`.

## Syntax

```
sgx_status_t sgx_sha384_close(
    sgx_sha_state_handle_t sha_handle
);
```

## Parameters

### **sha\_handle [in]**

This is a handle to the context state used by the cryptography library to perform an iterative SHA384 hash. The algorithm stores the intermediate results of performing the hash calculation over data sets.

## Return value

**SGX\_SUCCESS**

The SHA384 state was deallocated successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

The input handle is NULL.

**Description**

Calling `sgx_sha384_close` is the last step after performing a SHA384 hash over multiple datasets. The caller uses this function to deallocate memory used to store the SHA384 calculation state.

**Requirements**

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**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 96 bits (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 in 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, 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 source pointer and AAD pointer are NULL.

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

### **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 96 bits (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 in 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\_in\_mac [in]**

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, 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 source pointer and AAD pointer are NULL.

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

### 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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

#### **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 recommended 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 step 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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

### **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 *not* 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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**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

### **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

**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]**

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

### **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

### 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 *p_ecc_handle
);
```

### Parameters

#### **p\_ecc\_handle [out]**

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**

`sgx_ecc256_open_context` is utilized to allocate 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 function using this cryptographic system is completed or any error occurs, `sgx_ecc256_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

### **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

`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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

### **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 [out]**

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 [out]**

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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**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(
    const sgx_ec256_private_t *p_private_b,
    const 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 [out]**

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`:  $\text{pubKeyA} = \text{privKeyA} * G$ , 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`:  $\text{pubKeyB} = \text{privKeyB} * G$ , 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:

$$\text{shareKeyA} = \text{privKeyA} * \text{pubKeyB} = \text{privKeyA} * \text{privKeyB} * G.$$

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:

$$\text{shareKeyB} = \text{privKeyB} * \text{pubKeyA} = \text{privKeyB} * \text{privKeyA} * G.$$

As the following equation is true  $\text{privKeyA} * \text{privKeyB} * G = \text{privKeyB} * \text{privKeyA} * G$ , the result of both calculations is the same, that is, the equation  $\text{shareKeyA} = \text{shareKeyB}$  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	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

### **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 [out]**

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

**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,
    const 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 private key.

**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. Or 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

**sgx\_ecdsa\_verify**

`sgx_ecdsa_verify` verifies the input digital signature with a given public key over an input dataset.

**Syntax**

```
sgx_status_t sgx_ecdsa_verify(
    const uint8_t *p_data,
    uint32_t data_size,
    const sgx_ec256_public_t *p_public,
    const sgx_ec256_signature_t *p_signature,
    uint8_t *p_result,
    sgx_ecc_state_handle_t ecc_handle
);
```

**Parameters****p\_data [in]**

Pointer to the signed dataset to verify.

**data\_size [in]**

Size of the dataset to have its signature verified.

**p\_public [in]**

Pointer to the public key to be used in the calculation of the signature.

**NOTE:**

Value is LITTLE ENDIAN.

**p\_signature [in]**

Pointer to the signature to be verified.

**NOTE:**

Value is LITTLE ENDIAN.

**p\_result [out]**

Pointer to the result of the verification check populated by this function.

**ecc\_handle [in]**

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**

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**

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_EC_VALID` - Digital signature is valid

`SGX_EC_INVALID_SIGNATURE` - 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

**sgx\_rsa3072\_sign**

`sgx_rsa3072_sign` computes a digital signature for a given dataset based on RSA 3072 private key.

**Syntax**

```
sgx_status_t sgx_rsa3072_sign(
    const uint8_t *p_data,
    uint32_t data_size,
    const sgx_rsa3072_key_t *p_key,
    sgx_rsa3072_signature_t *p_signature
);
```

**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\_key [in]**

A pointer to the RSA key.

**NOTE:**

Value is LITTLE ENDIAN.

**p\_signature [out]**

A pointer to the signature generated by this function.

**NOTE:**

Value is LITTLE ENDIAN.

**Return value****SGX\_SUCCESS**

The digital signature is successfully generated.

**SGX\_ERROR\_INVALID\_PARAMETER**

The private key, data, 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 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 RSA 3072 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.

The scheme used for computing a digital signature is of the RSASSA-PKCS1-v1\_5 scheme.

#### Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

### **sgx\_rsa3072\_sign\_ex**

`sgx_rsa3072_sign_ex` computes a digital signature for a given dataset based on the RSA 3072 private key and the optional corresponding RSA 3072 public key.

#### Syntax

```
sgx_status_t sgx_rsa3072_sign_ex(
    const uint8_t *p_data,
    uint32_t data_size,
    const sgx_rsa3072_key_t *p_key,
    const sgx_rsa3072_public_key_t *p_public,
    sgx_rsa3072_signature_t *p_signature
);
```

## 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\_key [in]**

A pointer to the RSA private key.

---

#### **NOTE:**

Value is LITTLE ENDIAN.

---

### **p\_public [in]**

A pointer to the RSA public key. Can be NULL.

---

#### **NOTE:**

Value is LITTLE ENDIAN.

---

### **p\_signature [out]**

A pointer to the signature generated by this function.

---

#### **NOTE:**

Value is LITTLE ENDIAN.

---

## Return value

### **SGX\_SUCCESS**

The digital signature is successfully generated.

### **SGX\_ERROR\_INVALID\_PARAMETER**

The private key, data, or signature pointer is NULL. Or the data size is 0. Or the RSA private key and the public key do not match.

### **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 RSA 3072 private key and verifies the signature using the corresponding RSA 3072 public key if provided.

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.

The scheme used for computing a digital signature is of the RSASSA-PKCS1-v1\_5 scheme.

## Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

### sgx\_rsa3072\_verify

`sgx_rsa3072_verify` verifies the input digital signature for the given dataset based on the RSA 3072 public key.

### Syntax

```
sgx_status_t sgx_rsa3072_verify(
    const uint8_t *p_data,
    uint32_t data_size,
    const sgx_rsa3072_public_key_t *p_public,
    const sgx_rsa3072_signature_t *p_signature,
    sgx_rsa_result_t *p_result
);
```

### Parameters

#### **p\_data [in]**

A pointer to the signed dataset to be verified.

#### **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.

[Return value](#)

**SGX\_SUCCESS**

The digital signature verification was performed successfully. Check p\_result to get the verification result.

**SGX\_ERROR\_INVALID\_PARAMETER**

The 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 RSA 3072 public key.

A digital signature over a message is a buffer of 384-bytes, which could be created by function: `sgx_rsa3072_sign`. The scheme used for computing a digital signature is of the RSASSA-PKCS1-v1\_5 scheme.

The typical result of the digital signature verification is one of the two values:

`SGX_RSA_VALID` - Digital signature is valid

`SGX_RSA_INVALID_SIGNATURE` - Digital signature is not valid

[Requirements](#)

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

**sgx\_create\_rsa\_key\_pair**

`sgx_create_rsa_key_pair` generates public and private key pairs for the RSA cryptographic algorithm with input key size and extracts each part of the key pair to the prepared buffers.

**Syntax**

```
sgx_status_t sgx_create_rsa_key_pair(
    int n_byte_size,
    int e_byte_size,
    unsigned char *p_n,
    unsigned char *p_d,
    unsigned char *p_e,
    unsigned char *p_p,
    unsigned char *p_q,
    unsigned char *p_dmp1,
    unsigned char *p_dmql,
    unsigned char *p_iqmp);
```

**Parameters****n\_byte\_size [in]**

Size in bytes of the RSA key modulus.

**e\_byte\_size [in]**

Size in bytes of the RSA public exponent.

**p\_n [out]**

Pointer to the generated RSA modulus.

**p\_d [out]**

Pointer to the generated RSA private exponent.

**p\_e [in, out]**

Pointer to the generated RSA private exponent.

**p\_p [out]**

Pointer to the RSA key factor p.

**p\_q [out]**

Pointer to the RSA key factor q.

**p\_dmp1 [out]**

Pointer to the RSA key factor `dmp1`.

**p\_dmqr1 [out]**

Pointer to the RSA key factor `dmqr1`.

**p\_iqmp [out]**

Pointer to the RSA key factor `iqmp`.

**Return value****SGX\_SUCCESS**

RSA key pair is successfully generated.

**SGX\_ERROR\_INVALID\_PARAMETER**

Some of the pointers are NULL, or the input size is less than 0.

**SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory available to complete this operation.

**SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred during the RSA key pair generation.

**Description**

This function generates public and private key pairs for the RSA cryptographic algorithm and extracts each part of the key pair to the prepared buffers. If the RSA public exponent is specified, this function utilizes the specified RSA public exponent to the generated RSA key pair.

Before calling the function, you need to allocate memory for all the RSA key components (`n`, `d`, `e`, `p`, `q`, `dmp1`, `dmqr1`, `iqmp`).

**Requirements**

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**sgx\_create\_rsa\_priv1\_key**

`sgx_create_rsa_priv1_key` generates a private key for the RSA cryptographic algorithm with the input RSA key components (`n`, `e`, `d`).

## Syntax

```
sgx_status_t sgx_create_rsa_priv1_key(
    int n_byte_size,
    int e_byte_size,
    int d_byte_size,
    const unsigned char *le_n,
    const unsigned char *le_e,
    const unsigned char *le_d,
    void **new_pri_key1
);
```

## Parameters

### **n\_byte\_size [in]**

Size in bytes of the RSA key modulus.

### **e\_byte\_size [in]**

Size in bytes of the RSA public exponent.

### **d\_byte\_size [in]**

Size in bytes of the RSA private exponent.

### **le\_n [in]**

Pointer to the RSA key modulus buffer.

### **le\_e [in]**

Pointer to the RSA public exponent buffer. e.

### **le\_d [in]**

Pointer to the RSA private exponent buffer d.

### **new\_pri\_key1 [out]**

Pointer to the generated RSA private key.

## Return value

### **SGX\_SUCCESS**

RSA private key is successfully generated.

### **SGX\_ERROR\_INVALID\_PARAMETER**

Some of the pointers are NULL, or the input size is less than 0.

### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory available to complete this operation.

**SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred during the RSA private key generation.

**Description**

This function generates a private key for the RSA cryptographic algorithm with the input RSA key components (n, e, d).

**Requirements**

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

**sgx\_create\_rsa\_priv2\_key**

`sgx_create_rsa_priv2_key` generates a private key for the RSA cryptographic algorithm with the input RSA key components.

**Syntax**

```
sgx_status_t sgx_create_rsa_priv2_key(
    int mod_size,
    int exp_size,
    const unsigned char *p_rsa_key_e,
    const unsigned char *p_rsa_key_p,
    const unsigned char *p_rsa_key_q,
    const unsigned char *p_rsa_key_dmp1,
    const unsigned char *p_rsa_key_dmq1,
    const unsigned char *p_rsa_key_iqmp,
    void **new_pri_key2
);
```

**Parameters****mod\_size [in]**

Size in bytes of the RSA key modulus.

**exp\_size [in]**

Size in bytes of the RSA public exponent.

**p\_rsa\_key\_e [in]**

Pointer to the RSA public exponent buffer.

**p\_rsa\_key\_p [in]**

Pointer to the prime number p.

**p\_rsa\_key\_q [in]**

Pointer to the prime number  $q$ .

**p\_rsa\_key\_dmp1 [in]**

Pointer to the RSA factor  $dmp1$ .  $dmp1 = q \bmod (p-1)$

**p\_rsa\_key\_dmqr1 [in]**

Pointer to the RSA factor  $dmqr1$ .  $dmqr1 = p \bmod (q-1)$

**p\_rsa\_key\_iqmp [in]**

Pointer to the RSA factor  $iqmp$ .  $iqmp = q^{-1} \bmod p$

**new\_pri\_key2 [out]**

Pointer to the generated RSA private key.

**Return value****SGX\_SUCCESS**

RSA private key is successfully generated.

**SGX\_ERROR\_INVALID\_PARAMETER**

Some of the pointers are NULL, or the input size is less than 0.

**SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory available to complete this operation.

**SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred during the RSA private key generation.

**Description**

This function generates a private key for the RSA cryptographic algorithm with the input RSA key components ( $p$ ,  $q$ ,  $dmp1$ ,  $dmqr1$ ,  $iqmp$ ).

**Requirements**

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

### **sgx\_create\_rsa\_pub1\_key**

`sgx_create_rsa_pub1_key` generates a public key for the RSA cryptographic algorithm with the input RSA key components.

#### Syntax

```
sgx_status_t sgx_create_rsa_pub1_key(  
    int mod_size,  
    int exp_size,  
    const unsigned char *le_n,  
    const unsigned char *le_e,  
    void **new_pub_key1  
);
```

#### Parameters

##### **mod\_size [in]**

Size in bytes of the RSA key modulus.

##### **exp\_size [in]**

Size in bytes of the RSA public exponent.

##### **le\_n [in]**

Pointer to the RSA modulus buffer.

##### **le\_e [in]**

Pointer to the RSA public exponent buffer.

##### **new\_pub\_key1 [out]**

Pointer to the generated RSA public key.

#### Return value

##### **SGX\_SUCCESS**

RSA public key is successfully generated.

##### **SGX\_ERROR\_INVALID\_PARAMETER**

Some of the pointers are NULL, or the input size is less than 0.

##### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory available to complete this operation.

##### **SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred during the RSA public key generation.

## Description

This function generates a public key for the RSA cryptographic algorithm with the input RSA key components (n, e).

## Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

### **sgx\_free\_rsa\_key**

`sgx_free_rsa_key` cleans up and deallocates the input RSA key.

## Syntax

```
sgx_status_t sgx_free_rsa_key(
    void *p_rsa_key,
    sgx_rsa_key_type_t key_type,
    int mod_size,
    int exp_size
);
```

## Parameters

### **p\_rsa\_key [in]**

Pointer to the RSA key.

### **key\_type [in]**

RSA key type .

### **mod\_size[in]**

Size in bytes for the RSA key modules.

### **exp\_size[in]**

Size in bytes of the RSA public exponent.

## Return value

### **SGX\_SUCCESS**

RSA key is successfully cleaned up.

## Description

This function clears the RSA key generated by one of the following APIs:

[sgx\\_create\\_rsa\\_priv1\\_key](#)[sgx\\_create\\_rsa\\_priv2\\_key](#)[sgx\\_create\\_rsa\\_pub1\\_key](#)

You can use this function to deallocate the memory used for storing the RSA key.

## Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

### **sgx\_rsa\_pub\_encrypt\_sha256**

`sgx_rsa_pub_encrypt_sha256` performs the RSA-OAEP encryption operation with the SHA-256 algorithm.

### Syntax

```
sgx_status_t sgx_rsa_pub_encrypt_sha256(
    const void* rsa_key,
    unsigned char* pout_data,
    size_t* pout_len,
    const unsigned char* pin_data,
    const size_t pin_len
);
```

### Parameters

#### **rsa\_key [in]**

Pointer to the RSA public key.

#### **pout\_data [out]**

Pointer to the output cipher text buffer.

#### **pout\_len [out]**

Length of the output cipher text buffer.

#### **pin\_data [in]**

Pointer to the input data buffer.

#### **pin\_len [in]**

Length of the input data buffer.

### Return value

#### **SGX\_SUCCESS**

All the outputs are generated successfully.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

Some of the pointers are NULL, or the input data size is 0.

#### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory available to complete this operation.

#### **SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred during the encryption operation.

### Description

This function carries out the RSA-OAEP encryption scheme with the SHA256 algorithm to encrypt the input data stream of a variable length.

You should allocate the source, destination, and counter data buffers within the enclave.

### Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

#### **sgx\_rsa\_priv\_decrypt\_sha256**

`sgx_rsa_priv_decrypt_sha256` performs the RSA-OAEP decryption operation with the SHA-256 algorithm.

### Syntax

```
sgx_status_t sgx_rsa_priv_decrypt_sha256(
    const void* rsa_key,
    unsigned char* pout_data,
    size_t* pout_len,
    const unsigned char* pin_data,
    const size_t pin_len
);
```

### Parameters

**rsa\_key [in]**

Pointer to the RSA private key.

**pout\_data [out]**

Pointer to the output decrypted data buffer.

**pout\_len [out]**

Length of the output decrypted data buffer.

**pin\_data [in]**

Pointer to the input data buffer to be decrypted.

**pin\_len [in]**

Length of the input data buffer to be decrypted.

[Return value](#)

**SGX\_SUCCESS**

All the outputs are generated successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

Some of the pointers are NULL, or the input data size is 0.

**SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory available to complete this operation.

**SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred during the encryption operation.

[Description](#)

This function carries out the RSA-OAEP decryption scheme with the SHA256 algorithm to decrypt the input data stream of a variable length.

You should allocate the source, destination, and counter data buffers within the enclave.

[Requirements](#)

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

### **sgx\_calculate\_ecdsa\_priv\_key**

`sgx_calculate_ecdsa_priv_key` generates an ECDSA private key based on an input random seed.

#### Syntax

```
sgx_status_t sgx_calculate_ecdsa_priv_key(  
    const unsigned char* hash_drg,  
    int hash_drg_len,  
    const unsigned char* sgx_nistp256_r_m1,  
    int sgx_nistp256_r_m1_len,  
    unsigned char* out_key,  
    int out_key_len  
);
```

#### Parameters

##### **hash\_drg [in]**

Pointer to the input random seed.

##### **hash\_drg\_len [in]**

Length of the input random seed.

##### **sgx\_nistp256\_r\_m1 [in]**

Pointer to the buffer for n-1 where n is order of the ECC group used.

##### **sgx\_nistp256\_r\_m1\_len [in]**

Length for the buffer for nistp256.

##### **out\_key [out]**

Pointer to the generated ECDSA private key.

##### **out\_key\_len [in]**

Length of the prepared buffer for ECDSA private key.

#### Return value

##### **SGX\_SUCCESS**

ECDSA private key is successfully generated.

##### **SGX\_ERROR\_INVALID\_PARAMETER**

Some of the pointers are NULL, or the input size is less than 0.

##### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory available to complete this operation.

## **SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred during the ECDSA private key generation.

### Description

This function generates an ECDSA private key based on an input random seed.

### Requirements

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

## **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\_KDF\_MISMATCH**

Indicates the key derivation function does not match.

**SGX\_ERROR\_UNRECOGNIZED\_PLATFORM**

Intel® EPID Provisioning failed because the platform was not recognized by the back-end server.

**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	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

**sgx\_get\_ps\_sec\_prop\_ex**

`sgx_get_ps_sec_prop_ex` gets a data structure describing the security property of the platform service with extended platform service information.

**Syntax**

```
sgx_status_t sgx_get_ps_sec_prop_ex (
    sgx_ps_sec_prop_desc_ex_t* security_property
);
```

**Parameters****security\_property [out]**

A pointer to the buffer that receives the security property descriptor of the platform service and platform service information. 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	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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]**

The 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
- 0x3 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, in 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\_BUSY**

The requested service is temporarily not available.

**SGX\_ERROR\_MC\_OVER\_QUOTA**

The enclave has reached the quota 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.

Creating a monotonic counter (MC) involves writing to the non-volatile memory available in the platform. Repeated write operations could cause the

memory to wear out during the normal lifecycle of the platform. Intel® SGX prevents this by limiting the rate at which MC operations can be performed. If you exceed the limit, the MC operation may return `SGX_ERROR_BUSY` for several minutes.

Intel® SGX limits the number of monotonic counters (MC) an enclave can create. To avoid exhausting the available quota, an Intel SGX application should record the MC UUID that `sgx_create_monotonic_counter_ex` returns and destroy a MC when it is not needed any more. If an enclave reaches its quota and previously created MC UUIDs have not been recorded, you may restore the MC service after uninstalling the Intel® SGX PSW and installing it again. This procedure deletes all MCs created by any enclave in that system.

---

### **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	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

### **sgx\_create\_monotonic\_counter**

`sgx_create_monotonic_counter` creates a monotonic counter with default owner policy and default user attribute mask.

### 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\_BUSY**

The requested service is temporarily not available.

##### **SGX\_ERROR\_MC\_OVER\_QUOTA**

The enclave has reached the quota 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 the default owner policy `0x1`, which means enclaves with same signing key can access the monotonic counter and default `owner_attribute_mask` `0xFFFFFFFFFFFFFFFFCB`.

The caller should call `sgx_create_pse_session` to establish a session with the platform service enclave before calling this API.

Creating a monotonic counter (MC) involves writing to the non-volatile memory available in the platform. Repeated write operations could cause the memory to wear out during the normal lifecycle of the platform. Intel® SGX prevents this by limiting the rate at which MC operations can be performed. If you exceed the limit, the MC operation may return `SGX_ERROR_BUSY` for several minutes.

Intel® SGX limits the number of MCs an enclave can create. To avoid exhausting the available quota, an Intel SGX application should record the MC UUID that `sgx_create_monotonic_counter` returns and destroy a MC when it is not needed any more. If an enclave reaches its quota and previously created MC UUIDs have not been recorded, you may restore the MC service after uninstalling the Intel® SGX PSW and installing it again. This procedure deletes all MCs created by any enclave in that system.

---

### **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	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

### **sgx\_destroy\_monotonic\_counter**

`sgx_destroy_monotonic_counter` destroys a monotonic counter created by `sgx_create_monotonic_counter` or `sgx_create_monotonic_counter_ex`.

### Syntax

```
sgx_status_t sgx_destroy_monotonic_counter(
    const sgx_mc_uuid_t * counter_uuid
);
```

## Parameters

### **counter\_uuid [in]**

The monotonic counter ID to be destroyed.

## Return value

### **SGX\_SUCCESS**

Monotonic counter is destroyed successfully.

### **SGX\_ERROR\_INVALID\_PARAMETER**

Any of the pointers is invalid.

### **SGX\_ERROR\_BUSY**

The requested service is temporarily not available.

### **SGX\_ERROR\_MC\_NOT\_FOUND**

The Monotonic Counter does not exist or has been invalidated.

### **SGX\_ERROR\_MC\_NO\_ACCESS\_RIGHT**

The enclave does not 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.

`sgx_destroy_monotonic_counter` fails if the calling enclave does not match the owner policy and the attributes specified in the call that created the monotonic counter.

Destroying a Monotonic Counter (MC) involves writing to the non-volatile memory available in the platform. Repeated write operations could cause the memory to wear out during the normal lifecycle of the platform. Intel® SGX prevents this by limiting the rate at which MC operations can be performed. If you exceed the limit, the MC operation may return `SGX_ERROR_BUSY` for several minutes.

### Requirements

Header	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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\_BUSY**

The requested service is temporarily not available.

##### **SGX\_ERROR\_MC\_NOT\_FOUND**

The Monotonic Counter does not exist or has been invalidated.

##### **SGX\_ERROR\_MC\_NO\_ACCESS\_RIGHT**

The enclave does not 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.

`sgx_increment_monotonic_counter` fails if the calling enclave does not match the owner policy and the attributes specified in the call that created the monotonic counter.

Incrementing a monotonic counter (MC) involves writing to the non-volatile memory available in the platform. Repeated write operations could cause the memory to wear out during the normal lifecycle of the platform. Intel® SGX prevents this by limiting the rate at which MC operations can be performed. If you exceed the limit, the MC operation may return `SGX_ERROR_BUSY` for several minutes.

### Requirements

Header	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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 does not exist or has been invalidated.

### **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.

`sgx_read_monotonic_counter` fails if the calling enclave does not match the owner policy and the attributes specified in the call that created the monotonic counter.

## Requirements

Header	<code>sgx_tae_service.h</code> <code>sgx_tae_service.edl</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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 must 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_proc_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 occurred.

**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\_init\_ex**

The `sgx_ra_init_ex` function creates a context for the remote attestation and key exchange process while it allows the use of a custom defined Key Derivation Function (KDF).

**Syntax**

```
sgx_status_t sgx_ra_init_ex(
    const sgx_ec256_public_t * p_pub_key,
    int b_pse,
    sgx_ra_derive_secret_keys_t derive_key_cb,
    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 must 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.

### **derive\_key\_cb [in]**

This a pointer to a call back routine matching the function prototype of `sgx_ra_derive_secret_keys_t`. This function takes the Diffie-Hellman shared secret as input to allow the ISV enclave to generate their own derived shared keys (SMK, SK, MK and VK).

### **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_proc_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 occurred.

### [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	<code>sgx_tkey_exchange.h</code> <code>sgx_tkey_exchange.edl</code>
Library	<code>sgx_tkey_exchange.lib</code>

### 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_proc_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` or `SGX_RA_KEY_SK`.

If the RA context was generated by `sgx_ra_init`, the returned `SGX_RA_KEY_MK` or `SGX_RA_KEY_SK` is derived from the Diffie-Hellman shared secret elliptic curve field element between the service provider and the application enclave using the following Key Derivation Function (KDF):

$$KDK = \text{AES-CMAC}(\text{key0}, \text{gab x-coordinate})$$

$$\text{SGX\_RA\_KEY\_MK} = \text{AES-CMAC}(KDK, 0x01 || 'MK' || 0x00 || 0x80 || 0x00)$$

$$\text{SGX\_RA\_KEY\_SK} = \text{AES-CMAC}(KDK, 0x01 || 'SK' || 0x00 || 0x80 || 0x00)$$

The `key0` used in the key extraction operation is 16 bytes of `0x00`. The plain text used in the Key derivation calculation is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format. The plain text used in each key calculation includes:

- a counter (0x01)
- a label: the ASCII representation of one of the strings 'MK' or 'SK' in Little Endian format
- a bit length (0x80)

If the RA context was generated by the `sgx_ra_init_ex` API, the KDF used to generate `SGX_RA_KEY_MK` and `SGX_RA_KEY_SK` is defined in the implementation of the call back function provided to the `sgx_ra_init_ex` function.

### **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	<code>sgx_tkey_exchange.h</code> <code>sgx_tkey_exchange.edl</code>
Library	<code>sgx_tkey_exchange.lib</code>

### **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	<code>sgx_tkey_exchange.h</code> <code>sgx_key_exchange.edl</code>
Library	<code>sgx_tkey_exchange.lib</code>

### **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	<code>sgx_dh.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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	<code>sgx_dh.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

#### **sgx\_dh\_initiator\_proc\_msg1**

The initiator of the DH secure session establishment handles `msg1` sent by a responder, generates `msg2`, and records the ECC key pair of the initiator in the DH session structure.

---

**NOTE**

To use DH key exchange 2.0 APIs, define SGX\_USE\_LAv2\_INITIATOR.

---

**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]**

Pointer to the dh message 1 buffer generated by a session responder. The buffer must be in enclave address space.

---

**NOTE**

The pointer value must be a valid address within an enclave, as well as the end address of the session structure.

---

**msg2 [out]**

Pointer to the dh message 2 buffer. The buffer must be in enclave address space.

---

**NOTE**

The pointer value must be a valid address within an enclave, as well as the end address of the session structure.

---

**dh\_session [in/out]**

Pointer to the dh session structure used during establishment. The buffer must be in enclave address space.

---

**NOTE**

The pointer value 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 an incorrect order or state.

### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Enclave is out of memory.

### **SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred.

## Requirements

Header	sgx_dh.h
Library	sgx_tservice.lib or sgx_tservice_sim.lib (simulation)

### **sgx\_dh\_responder\_proc\_msg2**

Handles msg2 sent by an initiator, derives AEK, updates the session information, and generates msg3.

---

#### **NOTE**

To use DH key exchange 2.0 APIs, define SGX\_USE\_LAv2\_INITIATOR.

---

## 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]**

Pointer to the dh message 2 buffer generated by a session initiator. The buffer must be in enclave address space.

---

#### **NOTE**

---

---

The pointer value must be a valid address within an enclave, as well as the end address of the session structure.

---

### **msg3 [out]**

Pointer to the dh message 3 buffer generated by a session responder in this function. The buffer must be in enclave address space.

---

#### **NOTE**

The pointer value must be a valid address within an enclave, as well as the end address of the session structure.

---

### **dh\_session [in/out]**

Pointer to the dh session structure used during establishment. The buffer must be in enclave address space.

---

#### **NOTE**

The pointer value must be a valid address within an enclave, as well as the end address of the session structure.

---

### **aek [out]**

Pointer to instance of `sgx_key_128bit_t`. The aek is derived as follows:

```
KDK := CMAC(key0, LittleEndian(gab x-coordinate))
```

```
AEK = AES-CMAC(KDK, 0x01 || 'AEK' || 0x00 || 0x80 || 0x00)
```

The `key0` used in the key extraction operation is 16 bytes of `0x00`. The plain text used in the AES-CMAC calculation of the KDK is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format.

The plain text used in the AEK calculation includes:

- a counter (`0x01`)
  - a label: the ASCII representation of the string 'AEK' in Little Endian format)
  - a bit length (`0x80`)
- 

#### **NOTE**

The pointer value must be a valid address within an enclave, as well as the end address of the session structure.

---

### **initiator\_identity [out]**

Pointer to instance of `sgx_dh_session_enclave_identity_t`. Identity information of initiator includes `isv_svn`, `isv_product_id`, the enclave attributes, `MRSIGNER`, and `MRENCLAVE`. The buffer must be located in the enclave address space. Check the identity of the peer and decide whether to trust the peer and use the `aek`.

---

### **NOTE**

The pointer value 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 an incorrect order or state.

#### **SGX\_ERROR\_KDF\_MISMATCH**

Key derivation function does not match.

#### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Enclave is out of memory.

#### **SGX\_ERROR\_UNEXPECTED**

Unexpected error occurred.

### Requirements

Header	<code>sgx_dh.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (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 as follows:

```
KDK := CMAC(key0, LittleEndian(gab x-coordinate))
```

```
AEK = AES-CMAC(KDK, 0x01 || 'AEK' || 0x00 || 0x80 || 0x00)
```

The `key0` used in the key extraction operation is 16 bytes of `0x00`. The plain text used in the AES-CMAC calculation of the KDK is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format.

The plain text used in the AEK calculation includes:

- a counter (0x01)
- a label: the ASCII representation of the string 'AEK' in Little Endian format
- a bit length (0x80)

---

#### **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, the enclave attributes, MRSIGNER, and MRENCLAVE. 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	<code>sgx_dh.h</code>
Library	<code>sgx_tservice.lib</code> or <code>sgx_tservice_sim.lib</code> (simulation)

### **sgx\_fopen**

The `sgx_fopen` function creates or opens a protected file.

### Syntax

```
SGX_FILE* sgx_fopen (
```

```

    const char* filename,
    const char* mode,
    const sgx_key_128bit_t *key
);

```

## Parameters

### filename [in]

The name of the file to be created or opened.

### mode [in]

The file open mode string. Allowed values are any combination of 'r', 'w' or 'a', with possible '+' and possible 'b' (since string functions are currently not supported, 'b' is meaningless).

### key [in]

The encryption key of the file. This key is used as a key derivation key, used for deriving encryption keys for the file. If the file is created with `sgx_fopen`, you should protect this key and provide it as input every time the file is opened.

## Return value

If the function succeeds, it returns a valid file pointer, which can be used by all the other functions in the Protected FS API, otherwise, `NULL` is returned and `errno` is set with an appropriate error code. See Protected FS Error Codes for more details about errors.

## Description

`sgx_fopen` is similar to the C file API `fopen`. It creates a new Protected File or opens an existing Protected File created with a previous call to `sgx_fopen`. Regular files cannot be opened with this API.

For more details about this API and its parameters, check the `fopen` documentation.

## Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

### sgx\_fopen\_auto\_key

The `sgx_fopen_auto_key` function creates or opens a protected file.

## Syntax

```

SGX_FILE* sgx_fopen_auto_key(
    const char* filename,
    const char* mode
);

```

### Parameters

#### filename [in]

The name of the file to be created or opened.

#### mode [in]

The file open mode string. Allowed values are any combination of 'r', 'w' or 'a', with possible '+' and possible 'b' (since string functions are currently not supported, 'b' is meaningless).

### Return value

If the function succeeds, it returns a valid file pointer, which can be used by all the other functions in the Protected FS API, otherwise, `NULL` is returned and `errno` is set with an appropriate error code. See Protected FS Error Codes for more details about errors.

### Description

`sgx_fopen_auto_key` is similar to the C file API `fopen`. It creates a new Protected File or opens an existing Protected File created with a previous call to `sgx_fopen_auto_key`. Regular files cannot be opened with this API.

For more details about this API and its parameters, check the `fopen` documentation.

### Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

### sgx\_fclose

The `sgx_fclose` function closes a protected file handle.

### Syntax

```

int32_t sgx_fclose(
    SGX_FILE* stream
);

```

### Parameters

**stream [in]**

A file handle that is returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

**Return value**

0

The file was closed successfully.

1

There were errors during the operation.

**Description**

`sgx_fclose` is similar to the C file API `fclose`. It closes an open Protected File handle created with a previous call to `sgx_fopen` or `sgx_fopen_auto_key`. After a call to this function, the handle is invalid even if an error is returned.

For more details about this API and its parameters, check the `fclose` documentation.

**Requirements**

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

**sgx\_fread**

The `sgx_fread` function reads the requested amount of data from the file, and extends the file pointer by that amount.

**Syntax**

```
size_t sgx_fread(
    void* ptr,
    size_t size,
    size_t count,
    SGX_FILE* stream
);
```

**Parameters****ptr[out]**

A pointer to a buffer of at least `size*count` bytes, to receive the data read from the file.

**size [in]**

The size of each block to be read.

**count [in]**

The number of blocks to be read.

**stream [in]**

A file handle that is returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

**Return value**

The number of blocks of size `size` that were read from the file.

**Description**

`sgx_fread` is similar to the C file API `fread`. In case of an error, `sgx_ferror` can be called to get the error code.

For more details about this API and its parameters, check the `fread` documentation.

**Requirements**

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

**sgx\_fwrite**

The `sgx_fwrite` function writes the given amount of data to the file, and extends the file pointer by that amount.

**Syntax**

```
size_t sgx_fwrite(
    const void* ptr,
    size_t size,
    size_t count,
    SGX_FILE* stream
);
```

**Parameters****ptr [in]**

A pointer to a buffer of at least `size*count` bytes, that contains the data to write to the file

**size [in]**

The size of each block to be written.

**count [in]**

The number of blocks to be written.

**stream [in]**

A file handle that is returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

**Return value**

The number of blocks of size `size` that were written to the file.

**Description**

`sgx_fwrite` is similar to the C file API `fwrite`. In case of an error, `sgx_ferror` can be called to get the error code.

For more details about this API and its parameters, check the `fwrite` documentation.

**Requirements**

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

**sgx\_fflush**

The `sgx_fflush` function forces a cache flush, and if it returns successfully, it is guaranteed that your changes are committed to a file on the disk.

**Syntax**

```
int32_t sgx_fflush(
    SGX_FILE* stream
);
```

**Parameters****stream [in]**

A file handle that is returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

**Return value**

0

The operation completed successfully.

1

There were errors during the operation. `sgx_ferror` can be called to get the error code.

### Description

`sgx_fflush` is similar to the C file API `fflush`. This function flushes all the modified data from the cache and writes it to a file on the disk. In case of an error, `sgx_ferror` can be called to get the error code. Note that this function does not clear the cache, but only flushes the changes to the actual file on the disk. Flushing also happens automatically when the cache is full and page eviction is required.

For more details about this API and its parameters, check the `fflush` documentation.

### Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

### `sgx_ftell`

The `sgx_ftell` function creates or opens a protected file.

### Syntax

```
int64_t sgx_ftell(
    SGX_FILE* stream
);
```

### Parameters

#### **stream [in]**

A file handle that is returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

### Return value

If the function succeeds, it returns the current value of the position indicator of the file, otherwise, -1 is returned and `errno` is set with an appropriate error code. See Protected FS Error Codes for more details about errors.

### Description

`sgx_ftell` is similar to the C file API `ftell`.

For more details about this API and its parameters, check the `ftell` documentation.

### Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

### **sgx\_fseek**

The `sgx_fseek` function sets the current value of the position indicator of the file.

### Syntax

```
int64_t sgx_fseek(
    SGX_FILE* stream,
    int64_t offset,
    int origin
);
```

### Parameters

#### **stream [in]**

A file handle that was returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

#### **offset [in]**

The new required value, relative to the origin parameter.

#### **origin [in]**

The origin from which to calculate the offset (`SEEK_SET`, `SEEK_CUR` or `SEEK_END`).

### Return value

0

The operation completed successfully.

-1

There were errors during the operation. `sgx_ferror` can be called to get the error code.

### Description

`sgx_fseek` is similar to the C file API `fseek`.

For more details about this API and its parameters, check the `fseek` documentation.

### Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

### **sgx\_feof**

The `sgx_feof` function tells the caller if the file's position indicator hit the end of the file in a previous read operation.

### Syntax

```
int32_t sgx_feof(
    SGX_FILE* stream
);
```

### Parameters

#### **stream [in]**

A file handle that was returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

### Return value

0

End of file was not reached.

1

End of file was reached.

### Description

`sgx_feof` is similar to the C file API `feof`.

For more details about this API and its parameters, check the `feof` documentation.

### Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

**sgx\_ferror**

The `sgx_ferror` function returns the latest operation error code.

**Syntax**

```
int32_t sgx_ferror(
    SGX_FILE* stream
);
```

**Parameters****stream [in]**

A file handle that is returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

**Return value**

The latest operation error code is returned. 0 indicates that no errors occurred.

**Description**

`sgx_ferror` is similar to the C file API `ferror`. In case the latest operation failed because the file is in a bad state, `SGX_ERROR_FILE_BAD_STATUS` will be returned.

For more details about this API and its parameters, check the `ferror` documentation.

**Requirements**

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

**sgx\_clearerr**

The `sgx_clearerr` function attempts to repair a bad file status, and also clears the end-of-file flag.

**Syntax**

```
void sgx_clearerr(
    SGX_FILE* stream
);
```

**Parameters****stream [in]**

A file handle that is returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

### Return value

None

### Description

`sgx_clearerr` is similar to the C file API `clearerr`. This function attempts to repair errors resulted from the underlying file system, like write errors to the disk (resulting in a full cache that cannot be emptied). Call `sgx_ferror` or `sgx_feof` after a call to this function to learn if it was successful or not.

`sgx_clearerr` does not repair errors resulting from a corrupted file, like decryption errors, or from memory corruption, etc.

For more details about this API and its parameters, check the `clearerr` documentation.

### Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

### **sgx\_remove**

The `sgx_remove` function deletes a file from the file system.

### Syntax

```
int32_t sgx_remove(
    const char* filename,
);
```

### Parameters

#### **filename [in]**

The name of the file to delete.

### Return value

0

The operation completed successfully.

1

An error occurred, check `errno` for the error code.

## Description

`sgx_remove` is similar to the C file API `remove`.

For more details about this API and its parameters, check the `remove` documentation.

## Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

## **sgx\_fexport\_auto\_key**

The `sgx_fexport_auto_key` function is used for exporting the latest key used for the file encryption. See File Transfer with the Automatic Keys API for more details.

## Syntax

```
int32_t sgx_fexport_auto_key(
    const char* filename,
    sgx_key_128bit_t *key
);
```

## Parameters

### **filename [in]**

The name of the file to be exported. This should be the name of a file created with the `sgx_fopen_auto_key` API.

### **key [out]**

The latest encryption key.

## Return value

0

The operation completed successfully.

1

An error occurred, check `errno` for the error code.

## Description

`sgx_fexport_auto_key` is used to export the last key that was used in the encryption of the file. With this key you can import the file in a different enclave or system.

---

**NOTE:**

1. In order for this function to work, the file should not be opened in any other process.
  2. This function only works with files created with `sgx_fopen_auto_key`.
- 

See File Transfer with the Automatic Keys API for more details.

### Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

### `sgx_fimport_auto_key`

The `sgx_fimport_auto_key` function is used for importing a Protected FS auto key file created on a different enclave or platform. See File Transfer with the Automatic Keys API for more details.

### Syntax

```
int32_t sgx_fimport_auto_key(
    const char* filename,
    const sgx_key_128bit_t *key
);
```

### Parameters

#### **filename [in]**

The name of the file to be imported. This should be the name of a file created with the `sgx_fopen_auto_key` API, on a different enclave or system.

#### **key [in]**

The encryption key, exported with a call to `sgx_fexport_auto_key` in the source enclave or system.

### Return value

0

The operation completed successfully.

1

An error occurred, check `errno` for the error code.

### Description

`sgx_fimport_auto_key` is used for importing a Protected FS file. After this call returns successfully, the file can be opened normally with `sgx_fexport_auto_key`.

---

### NOTE:

1. In order for this function to work, the file should not be opened in any other process.
  2. This function only works with files created with `sgx_fopen_auto_key`.
- 

See File Transfer with the Automatic Keys API for more details.

### Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

### `sgx_fclear_cache`

The `sgx_fclear_cache` function is used for clearing the internal file cache. The function scrubs all the data from the cache, and releases all the allocated cache memory.

### Syntax

```
int32_t sgx_fclear_cache(
    SGX_FILE* stream
);
```

### Parameters

#### **stream [in]**

A file handle that is returned from a previous call to `sgx_fopen` or `sgx_fopen_auto_key`.

### Return value

0

The operation completed successfully.

1

An error occurred, call `sgx_ferror` to get the error code.

## Description

`sgx_fclear_cache` is used to scrub all the data from the cache and release all the allocated cache memory. If modified data is found in the cache, it will be written to the file on disk before being scrubbed.

This function is especially useful if you do not trust parts of your own enclave (for example, external libraries you linked against, etc.) and want to make sure there is as little sensitive data in the memory as possible before transferring control to the code they do not trust. Note, however, that the `SGX_FILE` structure itself still holds sensitive data. To remove all such data related to the file from memory completely, you should close the file handle.

## Requirements

Header	<code>sgx_tprotected_fs.h</code> <code>sgx_tprotected_fs.edl</code>
Library	<code>sgx_tprotected_fs.lib</code>

## `sgx_virtual_protect`

This function is used to change the access permissions for a committed page in an Intel® SGX 2.0 enclave.

## Syntax

```
int sgx_virtual_protect(
    void* addr,
    const size_t size,
    const uint32_t new_prot,
    const uint32_t* old_prot,
    sgx_status_t * error
);
```

## Parameters

### **addr [in]**

Page aligned address in the enclave that specifies the start of the address range on which to perform the operation.

### **size [in]**

The size of the address range on which to perform the operation, in bytes, as a multiple of page size.

### **new\_prot [in]**

Specifies the new page protections:

- `SGX_PAGE_NOACCESS`
- `SGX_PAGE_READONLY`
- `SGX_PAGE_READWRITE`
- `SGX_PAGE_EXECUTE_READ`
- `SGX_PAGE_EXECUTE_READWRITE`

**old\_prot [out]**

Optional. If passed in, the old page protections for the first page in the address range specified will be returned.

**error [out]**

Optional. If passed in, on failure, error will be set to provide more information.

Return value	Description
<code>SGX_ERROR_FEATURE_NOT_SUPPORTED</code>	This platform does not support Intel® SGX EDMM.
<code>SGX_ERROR_INVALID_PARAMETER</code>	One of the parameters passed in is invalid. This may be due to passing in a non page-aligned address, a size that is not a multiple of the page size, or an invalid permissions specifier.
<code>SGX_ERROR_INVALID_STATE</code>	The pages in the address range are not in the correct state for the requested operation.
<code>SGX_ERROR_MEMORY_LOCKED</code>	Special regions in the enclave are locked and unavailable for update by this function. Attempts to change locked regions will result in this error.
<code>SGX_ERROR_UNEXPECTED</code>	General run time error.

**Return value****0**

Indicates failure.

**nonzero**

Indicates success

**Description**

Intel® SGX 2.0 platform support is required. If the platform does not support Intel® SGX EDMM, `sgx_virtual_protect` will return 0 and set `error` to `SGX_ERROR_FEATURE_NOT_SUPPORTED`.

## Requirement

Header	sgx_tedmm.h, sgx_tedmm.edl
Library	sgx_tedmm.lib

### **sgx\_ecc256\_calculate\_pub\_from\_priv**

Generates an ECC public key based on a given ECC private key.

#### Syntax

```
sgx_ecc256_calculate_pub_from_priv(
    const sgx_ec256_private_t *p_att_priv_key,
    sgx_ec256_public_t *p_att_pub_key
);
```

#### Parameters

##### **p\_att\_priv\_key [in]**

Pointer to the input ECC private key.

##### **p\_att\_pub\_key [out]**

Pointer to output public key - LITTLE ENDIAN.

#### Return value

##### **SGX\_SUCCESS**

All outputs are generated successfully.

##### **SGX\_ERROR\_INVALID\_PARAMETER**

One or more of input parameters is invalid.

##### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory is available to complete this operation.

##### **SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failed.

#### Description

This function retrieves an ECC public key from a given private key on curve NID\_X9\_62\_prime256v1. (pub = priv \* curve\_group).

## Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

## sgx\_ecdsa\_verify\_hash

Directly verifies the signature for the given data of size SGX\_SHA256\_HASH\_SIZE based on the public key.

## Syntax

```
sgx_status_t sgx_ecdsa_verify_hash(
    const uint8_t *p_data,
    uint32_t data_size,
    const sgx_ec256_public_t *p_public,
    const sgx_ec256_signature_t *p_signature,
    uint8_t *p_result,
    sgx_ecc_state_handle_t ecc_handle
);
```

## Parameters

### p\_data [in]

Pointer to the signed dataset of size SGX\_SHA256\_HASH\_SIZE to be verified.

### p\_public [in]

Pointer to the public key to be used for the signature calculation.

---

#### **NOTE:**

Value is LITTLE ENDIAN.

---

### p\_signature [in]

Pointer to the signature to be verified.

---

#### **NOTE:**

Value is LITTLE ENDIAN.

---

### p\_result [out]

Pointer to the result of the verification check populated by this function.

### **ecc\_handle [in]**

Handle of the allocated and initialized ECC GF(p) context state used to call standard functions of the elliptic curve cryptosystem. The algorithm stores intermediate results of calculations performed using this context.

---

#### **NOTE:**

The ECC set of APIs only supports a 256-bit GF(p) cryptography system.

---

### Return value

#### **SGX\_SUCCESS**

Digital signature verification is 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**

Verification process failed due to an internal cryptography library error.

### Description

This function verifies the signature for the given data set based on the input public key. The function performs verification without calculating the data hash.

A digital signature of the message consists of a pair of large numbers, 256 bits each, which could be created by `sgx_ecdsa_sign`. The ECDSA scheme, an elliptic curve of the DSA scheme, is used for computing a digital signature.

The digital signature verification results in one of the following values:

`SGX_EC_VALID` - Digital signature is valid.

`SGX_EC_INVALID_SIGNATURE` - Digital signature is not valid.

To create elliptic curve domain parameters, use the `sgx_ecc256_open_context` function.

## Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

### sgx\_hmac\_sha256\_msg

Computes a message authentication code of p\_src using the hash function SHA256 and p\_key.

### Syntax

```
sgx_status_t sgx_hmac_sha256_msg(
    const unsigned char *p_src,
    int src_len,
    const unsigned char *p_key,
    int key_len,
    unsigned char *p_mac,
    int mac_len
);
```

### Parameters

#### **p\_src [in]**

Pointer to the input stream to be hashed.

#### **src\_len [in]**

Length in bytes of the input stream to be hashed.

#### **p\_key [in]**

Pointer to the key to be used in MAC operation.

#### **key\_len [in]**

Key length, in bytes.

#### **p\_mac [out]**

Pointer to the result MAC, must be allocated by the caller.

#### **mac\_len [in]**

Expected output MAC length, in bytes.

### Return value

**SGX\_SUCCESS**

All outputs are generated successfully.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

p\_src, p\_key, or p\_mac pointer is NULL.

src\_len, key\_len, or mac\_len size is less than or equal to 0.

#### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory is available to complete this operation.

#### **SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failed.

### Description

The function performs a standard HMAC hash over the input data buffer. Only a 256-bit version of the HMAC hash is supported.

Use this function if the complete input data stream is available. Otherwise, use the Init, Update... Update, Final procedure to compute the HMAC hash over multiple input data sets.

### Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

#### **sgx\_hmac256\_init**

Allocates and initializes the HMAC state to use p\_key.

### Syntax

```
sgx_status_t sgx_hmac256_init(
    const unsigned char *p_key,
    int key_len,
    sgx_hmac_state_handle_t *p_hmac_handle
);
```

### Parameters

**p\_key [in]**

Pointer to the key used in the message authentication operation.

**key\_len [in]**

Key length, in bytes.

**p\_hmac\_handle [out]**

Pointer to the output HMAC state handle.

**Return value**

**SGX\_SUCCESS**

All outputs are generated successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

p\_key or p\_hmac\_handle is NULL, or key\_len is less than or equal to 0.

**SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory is available to complete this operation.

**SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failed.

**Description**

Calling `sgx_hmac256_init` is the first step in performing the HMAC 256-bit hash over multiple data sets. Do not allocate memory for the HMAC state returned by this function. The state is specific to the implementation of the cryptography library, so the library performs the allocation itself. If the hash over the desired data sets is completed or any error occurs during the hash calculation process, call `sgx_hmac256_close` to free the state allocated by this algorithm.

**Requirements**

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**sgx\_hmac256\_update**

Authenticates chunks of a message during repetitive calls.

**Syntax**

```

sgx_status_t sgx_hmac256_update(
    const uint8_t *p_src,
    int src_len,
    sgx_hmac_state_handle_t hmac_handle
);

```

## Parameters

### **p\_src [in]**

Pointer to the input stream to be hashed.

### **src\_len [in]**

Length in bytes of the input stream to be hashed.

### **p\_hmac\_handle [in]**

Pointer to the HMAC state handle.

## Return value

### **SGX\_SUCCESS**

All outputs are generated successfully.

### **SGX\_ERROR\_INVALID\_PARAMETER**

p\_src or p\_hmac\_handle is NULL, or src\_len is less than or equal to 0.

### **SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failed.

## Description

Use this functions as a part of an HMAC 256-bit hash calculation over multiple data sets. For the HMAC hash calculation over a single data set, use the `sgx_hmac_sha256_msg` function instead. Before calling this function on the first data set, allocate and initialize the HMAC state structure, which will hold intermediate hash results, using the `sgx_hmac256_init` function. To obtain the hash after processing the final data set, call the `sgx_hmac256_final` function.

## Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

**sgx\_hmac256\_final**

Places the message authentication code in p\_hash.

**Syntax**

```
sgx_status_t sgx_hmac256_final(
    unsigned char *p_hash,
    int hash_len,
    sgx_hmac_state_handle_t hmac_handle
);
```

**Parameters****hash\_len [in]**

Expected MAC length, in bytes.

**hmac\_handle [in]**

Pointer to the HMAC state handle.

**p\_hash [out]**

Pointer to the resultant hash from the HMAC operation. This buffer should be allocated by the calling code.

**Return value****SGX\_SUCCESS**

All outputs are generated successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

p\_hash or hmac\_handle is NULL, or hash\_len is less than or equal to 0.

**SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failed.

**Description**

This function returns the hash after performing the HMAC 256-bit hash calculation over one or more data sets using the `sgx_hmac256_update`

function. Memory for the hash should be allocated by the calling code. The handle to the HMAC state used in the `sgx_hmac256_update` calls must be passed as input.

## Requirements

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

### **sgx\_hmac256\_close**

Cleans up the HMAC state.

#### Syntax

```
sgx_status_t sgx_hmac256_close(
    sgx_hmac_state_handle_t hmac_handle
);
```

#### Parameters

##### **p\_hmac\_handle [in]**

Pointer to the HMAC state handle.

#### Return value

##### **SGX\_SUCCESS**

HMAC state is cleaned up successful.

##### **SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failed.

#### Description

Calling `sgx_hmac256_close` is the last step after performing the HMAC hash over multiple data sets. Use this function to clean and deallocate memory used for storing the HMAC algorithm context state.

## Requirements

Header	sgx_tcrypto.h
Library	sgx_tcrypto.lib

**sgx\_aes\_gcm128\_enc\_init**

Returns an allocated and initialized AES-GCM encrypt algorithm context state. This should be part of the Init, Update ... Update, Final process when the AES-GCM encryption is to be performed over multiple datasets. If a complete dataset is available, you should call `sgx_rijndael128GCM_encrypt` to perform the encryption in a single call.

**Syntax**

```
sgx_status_t sgx_aes_gcm128_enc_init(
    const uint8_t *key,
    const uint8_t *iv,
    uint32_t iv_len,
    const uint8_t *aad,
    uint32_t aad_len,
    sgx_aes_state_handle_t* aes_gcm_state
);
```

**Parameters****key [in]**

Pointer to key to be used in the AES-GCM encryption operation. The size *must* be 128 bits.

**iv [in]**

Pointer to the initialization vector to be used in the AES-GCM calculation. NIST AES-GCM recommended IV size is 96 bits (12 bytes).

**iv\_len [in]**

Specifies the length of the input initialization vector. The length should be 12 as recommended by NIST.

**aad [in]**

Pointer to the additional authentication data buffer used in the GCM MAC calculation. The data in 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 the size can be zero.

### **aes\_gcm\_state [out]**

Handle to the context state used by the cryptography library to perform an iterative AES-GCM 128-bit encryption. The algorithm stores the intermediate results of performing the encryption over data sets.

### Return value

#### **SGX\_SUCCESS**

The AES-GCM encryption state is successfully allocated and initialized.

#### **SGX\_ERROR\_INVALID\_PARAMETER**

If key, MAC, or IV pointer is NULL.

If AAD size is > 0 and the AAD pointer is NULL.

The key or handle pointer is NULL.

#### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory is available to complete this operation.

#### **SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failure occurred.

### Description

Call `sgx_aes_gcm128_enc_init` as the first step in performing the AES-GCM encrypt over multiple datasets. Do not allocate memory for the AES-GCM 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 encryption over the desired datasets is completed or any error occurs during the encryption process, call `sgx_aes_gcm_close` to free the state allocated by this algorithm.

### Requirements

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**sgx\_aes\_gcm128\_enc\_update**

Performs AES-GCM 128-bit encryption over the input dataset provided. This function supports an iterative encryption over multiple datasets where `aes_gcm_handle` contains the intermediate results of the encryption over previous datasets.

**Syntax**

```
sgx_status_t sgx_aes_gcm128_enc_update (
    uint8_t *p_src,
    uint32_t src_len,
    uint8_t *p_dst,
    sgx_aes_state_handle_t aes_gcm_state
);
```

**Parameters****p\_src [in]**

Pointer to the input data stream to be encrypted.

**src\_len [in]**

Specifies the length on the input data stream to be encrypted.

**p\_dst [out]**

Pointer to the output cipher-text buffer.

**aes\_gcm\_state [in]**

Handle to the context state used by the cryptography library to perform AES-GCM encryption.

**Return value****SGX\_SUCCESS**

All the outputs are generated successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

The source pointer, destination pointer, or AES handle is NULL.

The source length is 0 or greater than `INT_MAX`.

**SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failure occurred while performing the AES-GCM encryption.

**NOTE:**

Unexpected errors indicate that the AES-GCM state is *not* freed. Call `sgx_aes_gcm_close` to free the AES-GCM state and avoid memory leak.

**Description**

This function encrypts data in the source input and puts it in `p_dst`. You should use it after initializing the AES-GCM state with `sgx_aes_gcm128_enc_init`.

**Requirements**

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**sgx\_aes\_gcm128\_enc\_get\_mac**

Obtains the authentication MAC from the given AES-GCM state.

**Syntax**

```
sgx_status_t sgx_aes_gcm128_enc_get_mac(
    uint8_t *mac,
    sgx_aes_state_handle_t aes_gcm_state
);
```

**Parameters****aes\_gcm\_state [in]**

Handle to the context state used by the cryptography library performing an iterative AES-GCM encryption.

**mac [out]**

Pointer to `SGX_AESGCM_MAC_SIZE` buffer to store MAC. The memory for the MAC should be allocated by the calling code.

**Return value****SGX\_SUCCESS**

The MAC is obtained successfully.

**SGX\_ERROR\_INVALID\_PARAMETER**

The MAC pointer or AES-GCM handle is NULL.

**SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failure occurred while performing the AES-GCM encryption.

**NOTE:**

If an unexpected error occurs, call `sgx_aes_gcm_close` to free the AES-GCM state to avoid memory leak.

**Description**

Writes `SGX_AESGCM_MAC_SIZE` bytes of the tag value to the buffer indicated by MAC.

**Requirements**

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

**sgx\_aes\_gcm\_close**

Cleans up and frees the AES-GCM state.

**Syntax**

```
sgx_status_t sgx_aes_gcm_close(
    sgx_aes_state_handle_t aes_gcm_state
);
```

**Parameters****aes\_gcm\_state [in]**

Pointer to the AES-GCM state handle.

**Return value****SGX\_SUCCESS**

The AES-GCM state was deallocated successfully.

**SGX\_ERROR\_UNEXPECTED**

Internal cryptography library failure occurred.

## Description

Call `sgx_aes_gcm_close` as the last step after performing AES-GCM over multiple datasets. Use this function to clean and deallocate the memory used for storing the AES-GCM algorithm context state.

## Requirements

Header	<code>sgx_tcrypto.h</code>
Library	<code>sgx_tcrypto.lib</code>

## 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_status_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_thread_sync_object_handle_t`
- `sgx_misc_select_t`
- `sgx_attributes_t`
- `sgx_misc_attribute_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_att_key_id_t`
- `sgx_ql_att_key_id_t`
- `sgx_att_key_id_ext_t`
- `sgx_qe_report_info_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_derive_secret_keys_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](#)
- [align\\_req\\_t](#)
- class template [custom\\_alignment\\_aligned](#)

### [sgx\\_enclave\\_id\\_t](#)

An enclave ID, also referred to as an enclave handle. Used as a handle to an enclave by various functions.

Enclave IDs are locally unique, i.e. within the platform, and the uniqueness is guaranteed until the next machine restart.

### Syntax

```
typedef uint64_t sgx_enclave_id_t;
```

### Requirements

Header	<a href="#">sgx_eid.h</a>
--------	---------------------------

### [sgx\\_status\\_t](#)

Specifies the return status from an Intel SGX function call. For a list containing all possible values of this data type, see Error Codes.

### Syntax

```
typedef enum _status_t { ... } sgx_status_t;
```

### Requirements

Header	<a href="#">sgx_error.h</a>
--------	-----------------------------

**sgx\_launch\_token\_t**

An opaque type used to hold enclave launch information. Used by [sgx\\_create\\_enclave](#) to initialize an enclave. The license is generated by the Launch Enclave.

See more details in [Loading and Unloading an Enclave](#).

**Syntax**

```
typedef uint8_t sgx_launch_token_t[1024];
```

**Requirements**

Header	sgx_urts.h
--------	------------

**sgx\_uswitchless\_worker\_type\_t**

Defines Switchless Calls worker thread type, trusted or untrusted. A worker can be either trusted (executed inside enclave) or untrusted (executed outside enclave).

**Syntax**

```
typedef enum {
    SGX_USWITCHLESS_WORKER_TYPE_UNTRUSTED,
    SGX_USWITCHLESS_WORKER_TYPE_TRUSTED
} sgx_uswitchless_worker_type_t;
```

**Requirements**

Header	sgx_uswitchless.h
--------	-------------------

**sgx\_uswitchless\_worker\_event\_t**

An application may register a callback to receive Switchless Calls events. The most useful information is presented by 4 worker events: a worker thread starts, a worker thread is idle, a worker thread missed some tasks, a worker thread exits.

**Syntax**

```
typedef enum {
    SGX_USWITCHLESS_WORKER_EVENT_START,
    SGX_USWITCHLESS_WORKER_EVENT_IDLE,
    SGX_USWITCHLESS_WORKER_EVENT_MISS,
    SGX_USWITCHLESS_WORKER_EVENT_EXIT,
}
```

```

    _SGX_USWITCHLESS_WORKER_EVENT_NUM,
} sgx_uswitchless_worker_event_t;

```

## Requirements

Header	sgx_uswitchless.h
--------	-------------------

### sgx\_uswitchless\_worker\_stats\_t

Switchless Calls gather statistics of calls processed by worker threads, and calls missed by worker threads and handled using fallback to regular ECALLs/OCALLs . An application can access the statistics values if it is registered to callbacks.

## Syntax

```

typedef struct {
    uint32_t processed;
    uint32_t missed;
} sgx_uswitchless_worker_stats_t;

```

## Members

### processed

32-bit counter that counts the number of tasks that all workers have processed.

### missed

32-bit counter that counts the number of tasks that all workers have missed.

## Requirements

Header	sgx_uswitchless.h
--------	-------------------

### sgx\_uswitchless\_worker\_callback\_t

Callback function that is called upon worker threads events and can be used to collect feature statistics and detect feature behavior for configuration tuning or other needs.

## Syntax

```

typedef void (*sgx_uswitchless_worker_callback_t) (
    sgx_uswitchless_worker_type_t type,
    sgx_uswitchless_worker_event_t event,

```

```
const sgx_uswitchless_worker_stats_t* stats);
```

## Parameters

### type

Worker thread type .

### event

Type of the event occurred.

### stats

Pointer to statistics data.

## Requirements

Header	sgx_uswitchless.h
--------	-------------------

### sgx\_uswitchless\_config\_t

Switchless Calls configuration structure passed to [sgx\\_create\\_enclave\\_ex](#) to select feature configuration.

## Syntax

```
typedef struct
{
    uint32_t switchless_calls_pool_size_qwords;
    uint32_t num_uworkers;
    uint32_t num_tworkers;
    uint32_t retries_before_fallback;
    uint32_t retries_before_sleep;
    sgx_uswitchless_worker_callback_t
    callback_func[_SGX_USWITCHLESS_WORKER_EVENT_NUM];
} sgx_uswitchless_config_t;
```

## Members

### switchless\_calls\_pool\_size\_qwords

Size of the Switchless Calls task pool (1 indicates a task pool of 64 tasks).

Default value: 1 (64 tasks)

Max value: 8 (512 tasks)

```
#define SL_DEFAULT_MAX_TASKS_QWORDS 1 //64
```

```
#define SL_MAX_TASKS_MAX_QWORDS 8 //512
```

### num\_uworkers

Number of untrusted worker threads that serve Switchless OCALLs.

### **num\_tworkers**

Number of trusted worker threads that serve Switchless ECALLs. This number is limited by TCSNum defined in an enclave configuration file. Exceeding the number of available TCS prevents several trusted worker threads from entering the enclave.

### **retries\_before\_fallback**

Number of retries client threads wait (assembly pause) for a worker thread to start executing a Switchless Call before falling back to a regular ECALL/OCALL.

Default value: 20000

```
#define SL_DEFAULT_FALLBACK_RETRIES 20000
```

### **retries\_before\_sleep**

Number of retries worker threads wait (assembly pause) on the Task Pool for an incoming Switchless Call request before the worker thread goes to sleep .

Default value: 20000

```
#define SL_DEFAULT_SLEEP_RETRIES 20000
```

### **Callback\_func**

Array of 4 callback functions for all event types. Optional, default value: NULL.

#### **Default Initialization**

At least one of `num_uworkers` or `num_tworkers` must not be 0. If both are 0, `sgx_create_enclave_ex` will return an error.

Other fields passed as 0 are replaced with the default field value.

A macro with default values provided.

```
#define SGX_USWITCHLESS_CONFIG_INITIALIZER {0, 1, 1, 0, 0, { 0 } }
```

It will be translated to `{1, 1, 1, 20000, 20000, { 0 } }`

#### **Requirements**

Header	<code>sgx_uswitchless.h</code>
--------	--------------------------------

### 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 instruction */
} sgx_exception_vector_t;
```

### Requirements

Header	<code>sgx_trts_exception.h</code>
--------	-----------------------------------

### 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	<code>sgx_trts_exception.h</code>
--------	-----------------------------------

### 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_cpu_context_t cpu_context;
    sgx_exception_vector_t exception_vector;
    sgx_exception_type_t exception_type;
} sgx_exception_info_t;
```

## Members

### cpu\_context

The context record that contains the processor context at the exception time.

### exception\_vector

The reason the exception occurs. This is the code generated by a hardware exception.

**exception\_type**

The exception type.

SGX\_EXCEPTION\_HARDWARE (3) indicates a HW exception.

SGX\_EXCEPTION\_SOFTWARE (6) indicates a SW exception.

**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.

**Return value****EXCEPTION\_CONTINUE\_SEARCH (0)**

The exception handler did not handle the exception and the RTS should call the next exception handler in the chain.

**EXCEPTION\_CONTINUE\_EXECUTION (-1)**

The exception handler handled the exception and the RTS should continue the execution of the enclave.

**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	<code>sgx_spinlock.h</code>
--------	-----------------------------

## `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	<code>sgx_thread.h</code>
--------	---------------------------

## `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\_thread\_sync\_object\_handle\_t**

Data type for Intel® SGX thread synchronization object handle.

**Syntax**

```
typedef struct _sgx_thread_sync_object_t *sgx_thread_sync_object_handle_t;
```

**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 parts of the XSAVE state, which corresponds to the xfrm value of attributes. Since the save state depends on the host system and the 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_EINITTOKEN_KEY 0x0000000000000020ULL	The enclave has access to a launch key
SGX_FLAGS_KSS 0x0000000000000080ULL	The enclave requires the KSS feature.

## xfrm

xfrm is a combination of the following values:

Value	Description
SGX_XFRM_LEGACY 0x0000000000000003ULL	FPU and Intel® Streaming SIMD Extensions states are saved
SGX_XFRM_AVX 0x0000000000000006ULL	Intel® Advanced Vector Extensions 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 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 a key request used for selecting the appropriate key and any additional parameters required in the derivation of the key. This is an 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;
    sgx_config_svn_t config_svn;
    uint8_t reserved2[434]
} sgx_key_request_t;
```

## Members

### key\_name

The key name requested. Possible values are below:

Key Name	Value	Description
SGX_KEYSELECT_EINITTOKEN	0x0000	Launch key
SGX_KEYSELECT_	0x0001	Provisioning key

PROVISION		
SGX_KEYSELECT_PROVISION_SEAL	0x0002	Provisioning seal key
SGX_KEYSELECT_REPORT	0x0003	Report key
SGX_KEYSELECT_SEAL	0x0004	Seal key

### key\_policy

Identify which inputs are required for the key derivation. Possible values are below:

Key policy name	Value	Description
SGX_KEYPOLICY_MRENCLAVE	0x0001	Derive key using the enclave's ENCLAVE measurement register
SGX_KEYPOLICY_MRSIGNER	0x0002	Derive key using the enclave's SIGNER measurement register
SGX_KEYPOLICY_NOISVPRODID	0x0004	Derive key without the enclave's ISVPRODID
SGX_KEYPOLICY_CONFIGID	0x0008	Derive key with the enclave's CONFIGID
SGX_KEYPOLICY_ISVFAMILYID	0x0010	Derive key with the enclave's ISVFAMILYID
SGX_KEYPOLICY_ISVEXTPRODID	0x0020	Derive key with the enclave's ISVEXTPRODID

### **NOTE**

If MRENCLAVE is used, that key can only be rederived by that particular enclave.

### 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

Attributes mask used to determine which enclave attributes must be included in the key. It only impacts the derivation of a seal key, a provisioning key, and a 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\_mask**

The misc mask used to determine which enclave misc select must be included in the key. Reserved for future function extension.

### **config\_svn**

The enclave CONFIGSVN field.

### **reserved2**

Reserved for future use. Must be set to zero.

### Requirements

Header	<code>sgx_key.h</code>
--------	------------------------

### [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	<code>sgx_report.h</code>
--------	---------------------------

### [sgx\\_mac\\_t](#)

This type is utilized as storage for the 128-bit CMAC value of the report data.

### Requirements

Header	<code>sgx_report.h</code>
--------	---------------------------

### 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	<code>sgx_report.h</code>
--------	---------------------------

### 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	<code>sgx_report.h</code>
--------	---------------------------

### sgx\_target\_info\_t

Data structure of report target information. This is an input to functions `sgx_create_report` and `sgx_init_quote`, which are used to identify the enclave (its measurement and attributes), which will be able to verify the generated REPORT.

### Syntax

```
typedef struct _target_info_t
{
    sgx_measurement_t mr_enclave;
    sgx_attributes_t attributes;
    uint8_t reserved1[2];
    sgx_config_svn_t config_svn;
    sgx_misc_select_t misc_select;
    uint8_t reserved2[8];
    sgx_config_id_t config_id;
    uint8_t reserved3[384];
} sgx_target_info_t;
```

## Members

### **mr\_enclave**

Enclave hash of the target enclave

### **attributes**

Attributes of the target enclave

### **reserved1**

Reserved for future use. Must be set to zero.

### **config\_svn**

Enclave CONFIGSVN.

### **misc\_select**

Misc select bits for the target enclave. Reserved for future function extension.

### **reserved2**

Reserved for future use. Must be set to zero.

### **config\_id**

Enclave CONFIGID

### **reserved3**

Reserved for future use. Must be set to zero.

## Requirements

Header	sgx_report.h
--------	--------------

### **sgx\_report\_body\_t**

Data structure that contains information about the enclave. This data structure is a part of the `sgx_report_t` structure.

### Syntax

```
typedef struct _report_body_t
{
    sgx_cpu_svn_t cpu_svn;
    sgx_misc_select_t misc_select;
    uint8_t reserved1[12];
    sgx_isvext_prod_id_t isv_ext_prod_id;
    sgx_attributes_t attributes;
    sgx_measurement_t mr_enclave;
}
```

```

uint8_t reserved2[32];
sgx_measurement_t mr_signer;
uint8_t reserved3[32];
sgx_config_id_t config_id;
sgx_prod_id_t isv_prod_id;
sgx_isv_svn_t isv_svn;
sgx_config_svn_t config_svn;
uint8_t reserved4[42];
sgx_isvfamily_id_t isv_family_id;
sgx_report_data_t report_data;
} sgx_report_body_t;

```

## Members

### **cpu\_svn**

Security version number of the host system TCB (CPU).

### **misc\_select**

Misc select bits for the target enclave. Reserved for future function extension.

### **reserved1**

Reserved for future use. Must be set to zero.

### **isv\_ext\_prod\_id**

ISV assigned Extended Product ID.

### **attributes**

Attributes for the enclave. See [sgx\\_attributes\\_t](#) for the definitions of these flags.

### **mr\_enclave**

Measurement value of the enclave.

### **reserved2**

Reserved for future use. Must be set to zero.

### **mr\_signer**

Measurement value of the public key that verified the enclave.

### **reserved3**

Reserved for future use. Must be set to zero.

### **config\_id**

The enclave CONFIGID.

### **isv\_prod\_id**

SV Product ID of the enclave.

### **isv\_svn**

ISV security version number of the enclave.

### **config\_svn**

CONFIGSVN field.

### **reserved4**

Reserved for future use. Must be set to zero.

### **isv\_family\_id**

ISV assigned Family ID.

### **report\_data**

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 Intel® 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;
    sgx_isv_svn_t pce_svn;
    uint32_t xeid;
    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 Intel® EPID signature type.

### **epid\_group\_id**

The Intel® EPID group id of the platform belongs to.

### **qe\_svn**

The svn of the QE.

### **pce\_svn**

The svn of the PCE.

### **extended\_epid\_group\_id**

The extended Intel® EPID group ID.

### **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\_att\_key\_id\_t

An opaque type which identifies the attestation key to use when generating a quote.

### Syntax

```
typedef struct _att_key_id_t {
    uint8_t att_key_id[256];
} sgx_att_key_id_t;
```

### Requirements

Header	sgx_quote.h
--------	-------------

### sgx\_qi\_att\_key\_id\_t

A single attestation key. Contains both QE identity and the attestation algorithm ID.

### Syntax

```
typedef struct _sgx_qi_att_key_id_t
{
    uint16_t id;
    uint16_t version;
    uint16_t mrsigner_length;
    uint8_t mrsigner[48];
    uint32_t prod_id;
    uint8_t extended_prod_id[16];
    uint8_t config_id[64];
    uint8_t family_id[16];
    uint32_t algorithm_id;
}sgx_qi_att_key_id_t;
```

### Members

#### id

Structure ID.

**version**

Structure version.

**mrsigner\_length**

Number of valid bytes in MRSIGNER.

**mrsigner**

SHA256 or SHA384 hash of the Public key that signed the QE. The lower bytes contain MRSIGNER. Bytes beyond mrsigner\_length are '0's.

**prod\_id**

Legacy Product ID of the QE.

**extended\_prod\_id**

Extended Product ID of the QE. All 0's for legacy format enclaves.

**config\_id**

Config ID of the QE.

**family\_id**

Family ID of the QE.

**algorithm\_id**

Identity of the attestation key algorithm.

**Requirements**

Header	sgx_quote.h
--------	-------------

**sgx\_att\_key\_id\_ext\_t**

An extended attestation key to use when generating a quote.

**Syntax**

```
typedef struct _sgx_att_key_id_ext_t
{
    sgx_q1_att_key_id_t base;
    uint8_t spid[16];
    uint16_t att_key_type;
    uint8_t reserved[80];
}
```

```
}sgx_att_key_id_ext_t;
```

## Members

### base

The base structure of `sgx_ql_att_key_id_t`.

### spid

Service provider ID for EPID quote. Should be 0s for ECDSA quote.

### att\_key\_type

For non-EPID quote, it should be 0. For EPID quote, it equals to `sgx_quote_sign_type_t`.

### reserved

The structure should have the same size as `sgx_att_key_id_t`.

## Requirements

Header	<code>sgx_quote.h</code>
--------	--------------------------

## `sgx_qe_report_info_t`

Data structure that contains the information from app enclave and report generated by Quoting Enclave. This is the input and output parameter from the `sgx_get_quote_ex` function.

## Syntax

```
typedef struct _qe_report_info_t
{
    sgx_quote_nonce_t nonce;
    sgx_target_info_t app_enclave_target_info;
    sgx_report_t qe_report;
} sgx_qe_report_info_t;
```

## Members

### nonce

The quote nonce from app enclave used to generate quote.

### app\_enclave\_target\_info

The target info of the app enclave used to generate quote.

**qe\_report**

The report generated by Quote Enclave.

**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 : Reserved

Bit 3 : Secure Message service

Bit 4-31 : 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\_ps\_sec\_prop\_desc\_ex\_t

Security property descriptor of platform service with extended platform service information.

### Syntax

```
typedef struct _ps_sec_prop_desc_ex
{
    sgx_ps_sec_prop_desc_t ps_sec_prop_desc;
    sgx_measurement_t pse_mrsigner;
    sgx_prod_id_t pse_prod_id;
    sgx_isv_svn_t pse_isv_svn;
} sgx_ps_sec_prop_desc_ex_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\_derive\_secret\_keys\_t

The `sgx_ra_derive_secret_keys_t` function should take the Diffie-Hellman shared secret as input to allow the ISV enclave to generate their own derived shared keys (SMK, SK, MK and VK). Implementation of the function should return the appropriate return value.

## Syntax

```
typedef sgx_status_t(*sgx_ra_derive_secret_keys_t)(
    const sgx_ec256_dh_shared_t* p_shared_key,
    uint16_t kdf_id,
    sgx_ec_key_128bit_t* p_smk_key,
    sgx_ec_key_128bit_t* p_sk_key,
    sgx_ec_key_128bit_t* p_mk_key,
    sgx_ec_key_128bit_t* p_vk_key
);
```

## Parameters

### **p\_shared\_key [in]**

The the Diffie-Hellman shared secret.

### **kdf\_id [in]**

Key Derivation Function ID.

### **p\_smk\_key [out]**

The output SMK.

### **p\_sk\_key [out]**

The output SK.

### **p\_mk\_key [out]**

The output MK.

### **p\_vk\_key [out]**

The output VK.

## Return value

**SGX\_SUCCESS**

Indicates success.

### **SGX\_ERROR\_INVALID\_PARAMETER**

Indicates an error that the input parameters are invalid.

### **SGX\_ERROR\_KDF\_MISMATCH**

Indicates key derivation function does not match.

### **SGX\_ERROR\_OUT\_OF\_MEMORY**

Not enough memory is available to complete this operation, or contexts reach the limits.

### **SGX\_ERROR\_UNEXPECTED**

Indicates that an unexpected error occurred.

#### Description

A pointer to a call back routine matching the function prototype.

#### Requirements

Header	sgx_tkey_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 Intel® 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;
    uint16_t quote_type;
    uint16_t kdf_id;
    sgx_ec256_signature_t sign_gb_ga;
    sgx_mac_t mac;
    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 (Little Endian)**

Indicates the quote type, linkable (1) or un-linkable (0).

### **kdf\_id (Little Endian)**

Key derivation function id.

### **sign\_gb\_ga (Little 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` or `sgx_ra_init_ex` 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` 2-byte TYPE, 2-byte KDF-ID, and `sign_gb_ga` using SMK as the AES-CMAC key. SMK is derived as follows:

```
KDK = AES-CMAC(key0, LittleEndian(gab x-coordinate))
```

```
SMK = AES-CMAC(KDK, 0x01 || 'SMK' || 0x00 || 0x80 || 0x00)
```

The `key0` used in the key extraction operation is 16 bytes of `0x00`. The plain text used in the AES-CMAC calculation of the KDK is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format.

The plain text used in the SMK calculation includes:

- a counter (`0x01`)
- a label: the ASCII representation of the string 'SMK' in Little Endian format
- a bit length (`0x80`)

If the ISV needs to use a different KDF than the default KDF used by Intel® SGX PSW, the ISV can use the `sgx_ra_init_ex` API to provide a callback function to generate the remote attestation keys used in the SIGMA protocol (SMK), verification (VK) and returned by the API `sgx_ra_get_keys` (SK, MK).

### **sig\_rl\_size**

Size of the `sig_rl`, in bytes.

### **sig\_rl**

Pointer to the Intel® EPID Signature Revocation List Certificate of the Intel® 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
{
    sgx_mac_t mac;
    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`, GID, and `quote[]`, using SMK. SMK is derived follows:

```
KDK = AES-CMAC(key0, LittleEndian(gab x-coordinate))
```

```
SMK = AES-CMAC(KDK, 0x01 || 'SMK' || 0x00 || 0x80 || 0x00)
```

The `key0` used in the key extraction operation is 16 bytes of `0x00`. The plain text used in the AES-CMAC calculation of the KDK is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format.

The plain text used in the SMK calculation includes:

- a counter (`0x01`)
- a label (the ASCII representation of the string 'SMK' in Little Endian format)
- a bit length (`0x80`)

If the ISV needs to use a different KDF than the default KDF used by Intel® SGX PSW, the ISV can use the `sgx_ra_init_ex` API to provide a callback function to generate the remote attestation keys used in the SIGMA protocol (SMK), verification (VK) and returned by the API `sgx_ra_get_keys` (SK, MK).

### **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:

```
KDK= AES-CMAC(key0, LittleEndian(gab x-coordinate))
```

```
VK = AES-CMAC(KDK, 0x01 || 'VK' || 0x00 || 0x80 || 0x00)
```

The `key0` used in the key extraction operation is 16 bytes of `0x00`. The plain text used in the AES-CMAC calculation of the KDK is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format.

The plain text used in the VK calculation includes:

- a counter (`0x01`)
- a label (the ASCII representation of the string 'VK' in Little Endian format)
- a bit length (`0x80`).

If the ISV needs to use a different KDF than the default KDF used by Intel® SGX PSW, the ISV can use the `sgx_ra_init_ex` API to provide a callback function to generate the remote attestation keys used in the SIGMA protocol (SMK), verification (VK) and returned by the API `sgx_ra_get_keys` (SK, MK).

### **Requirements**

Header	<code>sgx_key_exchange.h</code>
--------	---------------------------------

**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	<code>sgx_ukey_exchange.h</code>
--------	----------------------------------

**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	<code>sgx_ukey_exchange.h</code>
--------	----------------------------------

**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 101
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 Intel 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, report, 2-byte KDF-ID, and 0x00s using SMK as the AES-CMAC key. SMK is derived as follows:

```
KDK= AES-CMAC(key0, LittleEndian(gab x-coordinate))
SMK = AES-CMAC(KDK, 0x01 || 'SMK' || 0x00 || 0x80 || 0x00)
```

The key0 used in the key extraction operation is 16 bytes of 0x00. The plain text used in the AES-CMAC calculation of the KDK is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format.

The plain text used in the SMK calculation includes:

- a counter (0x01)
- a label: the ASCII representation of the string 'SMK' in Little Endian format
- a bit length (0x80)

## 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 as follows:

```
KDK = AES-CMAC(key0, LittleEndian(gab x-coordinate))
```

```
SMK = AES-CMAC(KDK, 0x01 || 'SMK' || 0x00 || 0x80 || 0x00)
```

The `key0` used in the key extraction operation is 16 bytes of `0x00`. The plain text used in the AES-CMAC calculation of the KDK is the Diffie-Hellman shared secret elliptic curve field element in Little Endian format.

The plain text used in the SMK calculation includes:

- a counter (0x01)
- a label: the ASCII representation of the string 'SMK' in Little Endian format
- a bit length (0x80)

### 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**

Intel 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_DISABLED_MANUAL_ENABLE,
    SGX_DISABLED_HYPERV_ENABLED,
    SGX_DISABLED_UNSUPPORTED_CPU
} 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.

### **SGX\_DISABLED\_MANUAL\_ENABLE**

Intel SGX device is disabled, but can be enabled manually in the BIOS setup.

### **SGX\_DISABLED\_HYPERV\_ENABLED**

An unsupported version of Windows\* 10 is detected with Hyper-V enabled.

### **SGX\_DISABLED\_UNSUPPORTED\_CPU**

Intel SGX is not supported by this CPU.

#### Requirements

Header	<code>sgx_capable.h</code>
--------	----------------------------

#### **sgx\_config\_svn\_t**

16-bits value representing the enclave CONFIGSVN. This value is used in the derivation of some keys.

#### Requirements

Header	<code>sgx_key.h</code>
--------	------------------------

#### **sgx\_config\_id\_t**

64-bytes value representing the enclave CONFIGID. This value is used in the derivation of some keys.

#### Requirements

Header	<code>sgx_key.h</code>
--------	------------------------

#### **sgx\_isvext\_prod\_id\_t**

16-bytes value representing the enclave Extended Product ID. This value is used in the derivation of some keys.

#### Requirements

Header	<code>sgx_report.h</code>
--------	---------------------------

#### **sgx\_isvfamily\_id\_t**

16-bytes value representing the enclave product Family ID. This value is used in the derivation of some keys.

#### Requirements

Header	<code>sgx_report.h</code>
--------	---------------------------

#### **sgx\_kss\_config\_t**

Structure of this type contains CONFIGSVN and CONFIGID values for a KSS enabled enclave. You can specify different CONFIGSVN and CONFIGID values

for the enclave to have additional control options over the key derivation process.

### Syntax

```
typedef struct _sgx_kss_config_t {
    sgx_config_id_t config_id;
    sgx_config_svn_t config_svn;
} sgx_kss_config_t;
```

### Members

#### **config\_id**

64-bytes value representing the enclave CONFIGID.

#### **config\_svn**

16-bits value representing the enclave CONFIGSVN.

### Requirements

Header	sgx_urts.h
--------	------------

### **align\_req\_t**

`align_req_t` is an offset-length pair used to describe the secrets within a structure.

### Syntax

```
typedef struct _req_data_t {
    size_t offset;
    size_t len;
} req_data_t;
```

### Requirements

Header	sgx_secure_align_api.h
--------	------------------------

### **custom\_alignment\_aligned**

`custom_alignment_aligned` is a class template used to align secrets that are statically-defined, for example, on the stack.

### Syntax

```
template<class T, std::size_t A, std::size_t... Ols>
class custom_alignment_aligned;
```

**T** is the class, structure, or type that needs alignment, for example, a structure representing or containing a cryptographic key.

**A** is the desired, traditional alignment of T. Do not confuse it with the alignment needed to mitigate the vulnerability - the two are related, but different.

**OLs** is a variable-length list of offset-length pairs. Each pair describes a secret within T. If T represents a single secret, there is only one pair, (0, sizeof(T)).

## Requirements

Header	sgx_secure_align.h
--------	--------------------

## Error Codes

Table 17 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_STATE	The API is invoked in incorrect order or state.
0x0007	SGX_ERROR_HYPERV_ENABLED	Incompatible versions of Windows* 10 OS and Hyper-V* are detected. Disable Hyper-V on the target system.
0x0008	SGX_ERROR_FEATURE_NOT_SUPPORTED	The feature is not supported.

0x1001	SGX_ERROR_INVALID_FUNCTION	The ECALL or 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. Possible reasons: <ul style="list-style-type: none"> <li>• ECALL is not public.</li> <li>• ECALL is blocked by the dynamic entry table.</li> <li>• A nested ECALL is not allowed during global initialization.</li> </ul>
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_ENCLAVE	The enclave is signed as a product enclave and cannot be created as a debuggable 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 the device.
0x2007	SGX_ERROR_MEMORY_MAP_	Page mapping failed in the driver.

	CONFLICT	
0x2009	SGX_ERROR_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 <code>sgx_sign</code> 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	Cannot open the enclave file.
0x2010	SGX_ERROR_INVALID_MISC	The MiscSelect or MiscMask settings are incorrect.
0x2012	SGX_ERROR_MEMORY_LOCKED	Attempt to change system memory that should not be modified.
0x3001	SGX_ERROR_MAC_MISMATCH	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 does 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	Intel® EPID blob verification error.
0x4004	SGX_ERROR_	Enclave has no privilege to get a launch token.

	SERVICE_INVALID_PRIVILEGE	
0x4005	SGX_ERROR_EPID_MEMBER_REVOKED	The Intel® EPID group membership has been revoked. The platform is not trusted. Updating the platform and repeating the operation will not remedy the revocation.
0x4006	SGX_ERROR_UPDATE_NEEDED	Intel® SGX requires update.
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 the 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.
0x4011	SGX_ERROR_KDF_MISMATCH	Key derivation function does not match during key exchange.
0x4012	SGX_ERROR_UNRECOGNIZED_PLATFORM	Intel® EPID Provisioning failed because the platform is not recognized by the back-end server.
0x4013	SGX_ERROR_SM_SERVICE_CLOSED	The secure message service instance has been closed.
0x4014	SGX_ERROR_SM_SERVICE_UNAVAILABLE	The secure message service applet does not have an existing session.
0x4015	SGX_ERROR_SM_SERVICE_UNCAUGHT_EXCEPTION	The secure message service instances terminated with an uncaught exception.

0x4016	SGX_ERROR_SM_SERVICE_RESPONSE_OVERFLOW	The response data of the service applet is too large.
0x4017	SGX_ERROR_SM_SERVICE_INTERNAL_ERROR	The secure message service got an internal error.
0x5002	SGX_ERROR_NO_PRIVILEGE	You do not have privilege to perform the operation.
0x6001	SGX_ERROR_PCL_ENCRYPTED	(Intel® SGX PCL) Trying to load an encrypted enclave using the wrong API or with wrong parameters
0x6002	SGX_ERROR_PCL_NOT_ENCRYPTED	(Intel® SGX PCL) Trying to load an enclave that is not encrypted using API or parameters for encrypted enclaves
0x7001	SGX_ERROR_FILE_BAD_STATUS	The file is in a bad status. Run <code>sgx_clearerr</code> to try and fix it.
0x7002	SGX_ERROR_FILE_NO_KEY_ID	The Key ID field is all zeros, cannot re-generate the encryption key.
0x7003	SGX_ERROR_FILE_NAME_MISMATCH	The current file name is different than the original file name (not allowed, substitution attack).
0x7004	SGX_ERROR_FILE_NOT_SGX_FILE	The file is not an Intel® SGX file.
0x7005	SGX_ERROR_FILE_CANT_OPEN_RECOVERY_FILE	A recovery file cannot be opened, so the flush operation cannot continue (only used when no EXXX is returned).
0x7006	SGX_ERROR_FILE_CANT_WRITE_RECOVERY_FILE	A recovery file cannot be written, so the flush operation cannot continue (only used when no EXXX is returned).
0x7007	SGX_ERROR_FILE_RECOVERY_NEEDED	When opening the file, recovery is needed, but the recovery process failed.
0x7008	SGX_ERROR_FILE_FLUSH_FAILED	fflush operation (to the disk) failed (only used when no EXXX is returned).
0x7009	SGX_ERROR_FILE_CLOSE_FAILED	fclose operation (to the disk) failed (only used when no EXXX is returned).
0x8001	SGX_ERROR_UNSUPPORTED_	Platform quoting infrastructure does not support the key.

	ATT_KEY_ID	
0x8002	SGX_ERROR_ATT_KEY_CERTIFICATION_FAILURE	Failed to generate and certify the attestation key.
0x8003	SGX_ERROR_ATT_KEY_UNINITIALIZED	The platform quoting infrastructure does not have the attestation key available to generate a quote.
0x8004	SGX_ERROR_INVALID_ATT_KEY_CERT_DATA	The data returned by the <code>sgx_get_quote_config()</code> of the platform library is invalid.
0x8005	SGX_ERROR_PLATFORM_CERT_UNAVAILABLE	The PCK Cert for the platform is not available.

## Container Support on Windows

Docker containers on Windows\* are supported on Windows\* Server 2019, Windows\* IOT Core 2019, and Windows\* IOT Enterprise 2019. This document will describe how to use Intel® SGX in Windows\* Nanoserver containers on Windows\* Server 2019, but the concepts / instructions can be generalized for the other host OSes and Docker containers.

### Windows\* Nano Server Installation

A quick set of powershell commands to set up your environment on Windows\* Server 2019 to support Docker containers are:

```
Install-Module -Name DockerMsftProvider -Repository  
PSGallery -Force
```

```
Install-Package -Name docker -ProviderName  
DockerMsftProvider
```

```
Restart-Computer -Force
```

A full description of the installation and configuration of the container environment is beyond the scope of this document. That information can be found in

<https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/set-up-environment>.

Once you have configured Docker, you can run a Windows\* Nano Server container by running the following command in powershell:

```
docker run -it --isolation=process  
mcr.microsoft.com/windows/nanoserver:1809
```

### Building a Windows\* Nano Server Test Application

To build applications that can be launched in Windows\* Nano Server, you need to make sure they are OneCore compliant. Enclave projects are already compatible with Windows\* Nano Server, but a typical Windows\* Console Application is not. The Windows\* Driver Kit (WDK) contains a project template "Application for Windows Drivers" that has the appropriate settings. These settings include generating code using /MT instead of /MD, as well as linking

against `OneCore.lib` and `ucrt.lib`, while ignoring `kernel32.lib`, `user-32.lib`, and `libucrt.lib`.

To set these manually, configure your application's project properties:

C/C++

Code Generation

Runtime Library: `/MT` for release, `/MTd` for debug

Linker

General

Additional Library Directories:

`$(WindowsSDK_LibraryPath)\$(PlatformTarget);`

`$(VC_LibraryPath_VC_x86_OneCore)` for 32bit or `$(VC_LibraryPath_VC_x64_OneCore)` for 64bit

[Debug] Input

Additional Dependencies:  
`OneCore.lib; ucrt[d].lib;`

Ignore Specific Default Libraries:  
`kernel32.lib; user-32.lib; libucrt[d].lib;`

The [QuoteGenerationSample](#) project is an example of a project configured to be OneCore compliant.

## Running the Test Application

If your test application is located in `c:\sgx_test`, you can mount this directory into your container by executing:

```
docker run -it --isolation=process -v c:\sgx_test:c:\sgx_test  
mcr.microsoft.com/windows/nanoserver:1809
```

Note that if the test application has any additional DLL dependencies, they must also be mounted or copied into the container.

If your application requires "Flexible Launch Control", first make sure that the Windows\* Server 2019 host has opted in to support this feature and your container must be launched with support for this device driver.

### Enabling Flexible Launch Control support on the host

In order to enable this feature on the host, add a `DWORD` named `SGX_Launch_Config_Optin` with value 1 to

```
HKEY_LOCAL_
MACHINE\SYSTEM\CurrentControlSet\Services\sgx_lc_
msr\Parameters
```

### Windows\* Nano Server with Flexible Launch Control support

In order to add the Flexible Launch Control device to the container, you must pass the `--device` option, and pass the GUID for the SGX Launch Token Interface:

```
--device="class/17eaf82e-e167-4763-b569-5b8273cef6e1"
```

For example,

```
docker run -it --isolation=process --
device="class/17eaf82e-e167-4763-b569-5b8273cef6e1"
mcr.microsoft.com/windows/nanoserver:1809
```

In Windows\*, using `--device` is only supported on containers that use process isolation. Containers that use hyperv isolation do not support this option.

### Putting it all together

If your application uses the Intel® SGX PSW as well as the Intel® SGX DCAP libraries, you could run on the host something like:

```
docker run -it --isolation=process -v c:\sgx\psw:c:\sgx_
psw -v c:\sgx\dcap:c:\sgx_dcap -v c:\sgx\test:c:\sgx_
test --device="class/17eaf82e-e167-4763-b569-5b8273ce-
f6e1" mcr.microsoft.com/windows/nanoserver:1809
```

Where `c:\sgx\psw` is a directory on the host OS that contains the Intel® SGX PSW, `c:\sgx\dcap` contains the Intel® SGX DCAP binaries, and `c:\sgx\test` contains the test collateral.

Once you are in the Windows\* Nano Server shell, you can run:

```
C:\>cd sgx_test
C:\sgx_test>mkdir c:\sgx_runtime
C:\sgx_test>copy c:\sgx_psw\sgx*dll c:\sgx_runtime
C:\sgx_test>set PATH=%PATH%;c:\sgx_runtime;c:\sgx_dcap
C:\sgx_test>.\App.exe
```

This step is optional: the default enclave load policy is persistent:

```
set the enclave load policy as persistent:succeed!
Step1: Call sgx_qe_get_target_info:succeed!
Step2: Call create_app_report:succeed!
Step3: Call sgx_qe_get_quote_size:succeed!
Step4: Call sgx_qe_get_quote:succeed!cert_key_type = 0x3
Clean up the enclave load policy:succeed!
```

There are other ways to copy/mount the necessary DLLs into the container, as well as setting the `PATH` environment variable. The steps above are merely one example. In addition, if you are using Intel® SGX DCAP, you should check whether your environment and application will need to use or will need to NOT use the `dcap_quoteprov.dll` provided by the Intel® SGX PSW when setting the `PATH` environment variable.

## **AESM Support for Containerized Client Applications**

### **AESM Configuration**

The traditional IPC mechanism between a client application and the AESM is not allowed to cross container boundaries. In order to support an IPC mechanism that is allowed to cross container boundaries, the AESM has the ability to open a TCP/IP socket and service client requests via that socket. The configuration for this service happens during AESM startup.

During startup, the AESM reads the IP address and port that it should attempt to bind to. If the registry entry does not exist or it is unable to bind to that address+port, the AESM will not be able to service requests via TCP/IP . If, for example, you would like the AESM to bind to all its interfaces on a port dynamically chosen by the OS, you would add the following registry entry:

```
reg add
HKLM\System\CurrentControlSet\Services\AESMService\State
\Sockets /v BIND_ADDR /d "0.0.0.0:0"
```

If you want to configure the AESM to only listen on a specific interface, such as the one associated with the IP address 10.1.2.3, and a particular port, for example, 5678, then you would add:

```
reg add
HKLM\System\CurrentControlSet\Services\AESMService\State
\Sockets /v BIND_ADDR /d "10.1.2.3:5678"
```

After restarting the AESM, the result of the bind operation is stored in ADDR (REG\_SZ) entry in the same key:

```
sc stop aesmservice

sc start aesmservice

reg query
HKLM\System\CurrentControlSet\Services\AESMService\State
\Sockets

HKEY_LOCAL_
MACHINE\System\CurrentControlSet\Services\AESMService\St
ate\Sockets

BIND_ADDR REG_SZ 0.0.0.0:0

ADDR REG_SZ 0.0.0.0:50123
```

You can confirm the AESM is listening on the port using netstat or any of the other traditional tools.

---

**NOTE**

If your AESM is configured to listen on a particular address:port, please make sure that your firewall will allow connections to this address:port pair from the client address:port.

---

### Client Application Configuration

For the client you will need to indicate that instead of the traditional IPC on the same host, you wish to have the client use TCP/IP socket IPC with the AESM, possibly running in a different container. You can do this by setting the following environment variable for the client application:

- `SGX_AESM_ADDR`: A resolvable host name or IP address and port pair on which the AESM is listening for new connections. For example,  
`set SGX_AESM_ADDR=127.0.0.1:12345`

Setting this environment variable directs the client application to use TCP/IP sockets for IPC with the AESM. If the client application is unable to connect to the AESM via TCP/IP, the client application will **not** fall back and try to connect via the other IPC mechanisms.

### AESM Inside Windows\* Nano Server

Another scenario users may be interested in is running the AESM inside a container. Once you are running inside a Windows\* Nano Server with a copy of the Intel® SGX PSW in `c:\sgx_psw` as described above, you can register the AESM as a service, and then start it as follows:

```
host>docker run -it --isolation=process v c:\SGX_
PSW:c:\SGX_PSW -u ContainerAdministrator
mcr.microsoft.com/windows/nanoserver:1809

cd c:\sgx_psw

reg add
HKLM\System\CurrentControlSet\Services\AESMSERVICE\State
\Sockets /v BIND_ADDR /d "0.0.0.0:0"

reg add
HKLM\SYSTEM\CurrentControlSet\Services\AESMSERVICE /v
InitHelper /t REG_DWORD /d 1

c:\sgx_psw>aesm_service.exe /service

c:\sgx_psw>sc start aesmservice
```

The `docker run` command is shown to explicitly demonstrate the need for running as an administrator in the container, since administrator rights are needed in order to register the Intel® SGX AESM service. You can verify the AESM has started and obtain the necessary parameters for the client application's configuration by executing the following commands inside the container:

```
sc query aesmservice  
  
ipconfig  
  
reg query  
HKLM\System\CurrentControlSet\Services\AESMService\State  
\Sockets
```

Afterwards, you can then configure your client application's environment variable as described in "Client Application Configuration."

## Security Issues Which May Require More Than Using the Latest Intel SGX SDK

This section lists security issues where mitigation may require more than using the latest Intel SGX SDK. The issues are listed in reverse chronological order.

Some of the issues are associated with the SW hardening needed attestation response.

### **INTEL-SA-00657, INTEL-SA-00767 (Addressing Stale Data Read from Legacy xAPIC)**

This section describes changes introduced in version 2.16.101.1 of the Intel SGX SDK for Windows.

[INTEL-SA-00657](#) describes an issue, named Stale Data Read from Legacy xAPIC, that affects SGX. As explained here in the associated technical paper, [the associated technical paper](#), Intel has provided a microcode update (MCU) to help mitigate the issue. However and as the paper explains, the best security for some enclaves may also require enclave SW changes. In the future, Intel expects to provide an additional MCU that will mitigate the issue such that enclave SW changes to help mitigate the issue will no longer be needed. This section describes:

1. changes in the Intel SGX SDK related to helping mitigate the issue.
2. changes that SGX developers may need to make to their enclaves, beyond rebuilding their enclaves with the new SDK, to achieve better security.

The SDK changes help ensure that enclave reads of memory outside the enclave are 8-byte-aligned and a multiple of 8 bytes in size (henceforth, safe reads). The new code does this through simple double-buffering where the destination of a safe read is a temporary buffer (the double-buffer). Then, this temporary buffer serves as the source of the original read. For example, suppose an enclave needs to read 100 bytes of memory outside the enclave, starting at address 0x10000001. One safe way to do this is for the enclave to read 104 (=8\*13) bytes in multiples of 8 bytes, starting at address 0x10000000, to a temporary buffer inside the enclave. Then, copy 100 bytes starting at the second byte (offset 1) of the temporary buffer to the original, non-temporary destination buffer. (Another safe way would recognize that the 88 bytes starting at 0x10000008 do not need to pass through a temporary buffer.)

The new SDK includes an updated Edger8r tool that generates code that does safe reads of memory outside the enclave. The updated SDK also has updated `memcpy` and `memcpy_s` functions. The updated versions check whether the source and/or destination buffers are outside the enclave and operate appropriately based on the results of these checks. (Checking the destination buffer is associated with the mitigation for [INTEL-SA-00615](#).) The updated SDK also introduces a `memcpy_nochecks` function that doesn't check where the buffers are. `memcpy_nochecks` can be used in cases where it's known that source and destination buffers are both inside the enclave.

For enclaves that read memory outside the enclave using code that isn't associated with ECALLs or OCALLs and for enclaves that use the EDL [`user_check`] attribute or that use nested pointers, the fact that the new Edger8r generates code with mitigations doesn't help. In these cases, SGX developers will still get [INTEL-SA-00657](#) mitigations if they use `memcpy` or `memcpy_s` to do the reads. Otherwise, developers of code like this may need to change their code to either mimic the behavior in the updated `memcpy` functions or convert assignment statements that cause reads of memory outside the enclave to `memcpy` or `memcpy_s` calls.

On affected processors, mitigation of these issues requires either

1. Intel microcode released February 2023
2. Disabling Hyper-Threading (if applicable)

Or

1. Intel microcode released August 2022
2. Disabling Hyper-Threading (if applicable)
3. SW mitigations described above

### **SW Hardening**

If the August 2022 microcode is used, then affected processors will get the SW hardening needed attestation response.

### **[INTEL-SA-00615 \(Mitigations for Processor MMIO Stale Data Vulnerabilities\)](#)**

[INTEL-SA-00615](#) describes four vulnerabilities, each with their own CVE. These four vulnerabilities are collectively known as Processor MMIO Stale Data vulnerabilities. The four CVEs are [CVE-2022-21123](#), [CVE-2022-21125](#), [CVE-2022-21127](#) and [CVE-2022-21166](#). [CVE-2022-21127](#) does not require SW

mitigations for SGX; the latest processor microcode alone is sufficient. The other CVEs require both processor microcode and SW to mitigate.

Because the Intel SGX security model does not trust the OS, a malicious OS could map MMIO memory into the untrusted memory space of an application that uses one or more Intel SGX enclaves. This could include the region of untrusted memory used for parameter passing to/from ECALLs and OCALLs, or any external buffers that an enclave might use to communicate with its application. If the malicious OS did such a mapping, then when the enclave wrote to this memory, it could propagate the stale data in its fill buffers into the uncore, where it could later be extracted by malicious software.

The mitigations below assume that Intel HT Technology is disabled to ensure that once the fill buffers are overwritten, a sibling thread cannot repopulate them. The necessary mitigation depends on how the enclave is accessing the non-enclave memory regions. See the [Processor MMIO Stale Data vulnerabilities technical paper](#) for more information.

Beginning with version 2.16.0.3, Intel has updated the Intel SGX SDK for Windows and the Edger8r tool included in the SDK to help prevent fill buffer data exposure through the code generated by the Edger8r tool. Similarly, the Intel SGX SDK now includes updates that will help prevent fill buffer data exposure through the code used by enclaves that use the switchless mode supported by the Intel SDK.

For enclaves that write to memory outside the enclave using code that isn't associated with ECALLs or OCALLs and for enclaves that use the EDL `[user_check]` attribute or that use nested pointers, SGX developers must add mitigations to their enclave source code to help prevent fill buffer data exposure. In order to mitigate, all writes to untrusted memory (that is, memory outside the enclave) must

1. either be preceded by the `VERW` instruction (with memory, not register, operand) and followed by the `MFENCE; LFENCE` instruction sequence or
2. must be in multiples of 8 bytes, aligned to an 8-byte boundary.

Specifically, for narrow (not a multiple of 8 bytes) or unaligned enclave writes to untrusted memory, the recommended mitigation is, for the narrow, one-byte case:

```
; rdi contains the write address outside the
enclave
```

```
SUBQ $8, %rsp ; assume stack exists and at least
8 bytes of stack available

MOVW %ds, (%rsp)

VERW (%rsp)

MOVB %al, (%rdi) ; narrow write to memory out-
side the enclave

MFENCE

LFENCE

ADDQ $8, %rsp
```

Note that the VERW instruction updates the ZF bit in the EFLAGS register, so exercise caution when using the above sequence in-line in existing code. Note also that the latest processor microcode has additions to VERW that are necessary for VERW to clear the fill buffers in the above sequence.

Mitigating a narrow or unaligned write to memory outside the enclave requires the instruction that does the write to be a single memory operand instruction; `MOVS` or `REP MOVS`, for example, cannot be used (unless the associated writes are all multiples of eight bytes and aligned). Read-modify-write instructions like `ADD` or even `CMPXCHG` can be treated like single memory operand write instructions, that is, add the mitigation sequence if the memory operand is narrow or unaligned. For example, for `CMPXCHG` with a 32-bit memory operand:

```
; rdi contains the address outside the enclave
; ecx contains the "new" value
; eax contains the "compare" value

SUBQ $8, %rsp ; assume stack exists and at least
8 bytes of stack available

MOVW %ds, (%rsp)

VERW (%rsp)

LOCK CMPXCHGL %ecx, (%rdi)

MFENCE

LFENCE
```

```
ADDQ $8, %rsp
```

The latest Intel SGX SDK includes new utility functions to facilitate SGX developers mitigating their code: `memcpy_verw`, `memcpy_verw_s`, `memmove_verw`, `memmove_verw_s`, `memset_verw` and `memset_verw_s`. These functions include width and alignment checks and will use the mitigation sequence as needed based on the results of these checks. They do not check whether the destination (write) address is outside the enclave; this is the responsibility of the calling code.

SGX developers should increment the ISVSVN values of their enclaves with the mitigations for Processor MMIO Stale Data Vulnerabilities ([INTEL-SA-00615](#)).

### SW Hardening

Affected processors will get the SW hardening needed attestation response.

### Addressing MXCSR Configuration Dependent Timing

[Data Operand Independent Timing ISA Guidance](#) describes MXCSR Configuration Dependent Timing (MCDT) where SW is needed to help ensure instruction timing that is independent of the values of the instruction's data operands.

If the intended operation of an Intel SGX enclave, for the lifetime of the enclave, is achievable with `MXCSR=0x1FBF`, then any loads of `MXCSR` (via `LDMXCSR`, `XRSTOR`, etc.) should be of `0x1FBF`. Enclaves that don't use SSE\* floating point instructions fall into this category. There should be an `LFENCE` between any load of `MXCSR` (even of `0x1FBF`) and subsequent use of any affected instruction (see [Data Operand Independent Timing ISA Guidance](#)).

If an enclave is compatible with `MXCSR=0x1FBF` and uses any of the affected instructions, then the beginning of each enclave `ECALL` should set `MXCSR` to `0x1FBF` and then execute an `LFENCE` instruction. The Intel SGX SDK has been changed to do this alleviating the need for SGX developers using the Intel SGX SDK to do so.

If the intended operation of an Intel SGX enclave is not achievable with `MXCSR=0x1FBF`, then the following mitigation sequence should be used:

```
; at this point, MXCSR != 0x1FBF
STMXCSR save_val
LDMXCSR value_0x1fbf
```

```

LFENCE
; Constant-time code using instructions affected
by MCDT
LFENCE
LDMXCSR save_val

```

### SW Hardening

This issue is not associated with the SW hardening needed attestation response.

### INTEL-SA-00334 (Enable CVE-2020-0551 Mitigation)

The Intel® SGX SDK facilitates mitigation of CVE-2020-0551, aka LVI (Load Value Injection). The Intel® SGX SDK supports two mitigation levels. One level addresses all instructions vulnerable to LVI. This level is called CVE-2020-0551-Load (Load, for short). The second mitigation level addresses vulnerable control flow instructions only and is called CVE-2020-0551-CF (CF, for short).

For more information on LVI, see [CVE-2020-0551](#) and [Intel's LVI tech paper](#).

### Mitigation enabled Trust Libraries

The Intel® SGX SDK includes three sets of trusted libraries: Unmitigated, Load and CF.

Mitigation Level	Path
Unmitigated	[Intel SGX SDK Install Path] bin\x64\Release
Load	[Intel SGX SDK Install Path]bin\x64\CVE-2020-0551-Load-Release
CF	[Intel SGX SDK Install Path]bin\x64\CVE-2020-0551-CF-Release

As stated above, the Intel® SGX SDK facilitates mitigation of CVE-2020-0551. To take full advantage of it, you need the following tools.

Required tools:

- Visual Studio v15.9.21 or higher (Visual Studio v15.x.y := Visual Studio 2017).
- Sgx-asm-pp.py. This is a custom tool that is a part of the Intel® SGX SDK. This tool helps mitigate handwritten assembly code.

Required options:

	Load mitigation level	CF mitigation level	No LVI mitigation
Visual Studio 15.9.21 or later	/Qspectre-load	/Qspectre-load-cf	(neither)
Sgx-asm-pp.py	--MITIGATION-CVE-2020-0551=LOAD	--MITIGATION-CVE-2020-0551=CF	--MITIGATION-CVE-2020-0551=NONE or use NASM/MASM directly.

Sgx-asm-pp.py has another required option, `--assembler`. This option specifies the assembler used for the assembly source file. Valid arguments for `--assembler` are 'nasm' and 'ml64' (without the quotes).

---

### **NOTE:**

To make sure the Binutils above are utilized in the mitigation build, you need to ensure the Binutils tools can be found before the system Binutils. The recommended action is to download the Binutils tools, copy them to `/usr/local/bin`, and make sure they have execute permission. For some OSs, setting `/usr/local/bin` does not work: for example, SUSE Linux Enterprise Server\*. You can utilize the `gcc -B` option to specify the Binutils path.

---

### **Create CVE-2020-0551 Mitigation enabled trusted (enclave) project**

You can use the Intel® SGX Visual Studio Wizard to create CVE-2020-0551 mitigation enabled new projects.

Follow [Using Microsoft\\* Visual Studio\\* Intel® Software Guard Extensions Wizard](#) to create a CVE-2020-0551 mitigation enabled new project.

After the new trusted project is created, four new CVE-2020-0551 mitigation related configurations will be found.

- CVE-2020-0551-Load-Release
- CVE-2020-0551-Load-Prerelease
- CVE-2020-0551-CF-Release
- CVE-2020-0551-CF-Prerelease

See [Mitigation Configuration Introduction](#) for detailed information on these configurations.

---

### **NOTE:**

---

---

These four configurations are only supported on x64 platform.

---

### Enable Mitigation for existing trusted project

User can use Intel® SGX Visual Studio Plug-In to add CVE-2020-0551 mitigation enabled configurations into existing trusted projects.

Follow [Using Microsoft\\* Visual Studio\\* Intel® Software Guard Extensions Add-in](#) to add these configurations.

After that, four new CVE-2020-0551 mitigation related configurations will be added.

- CVE-2020-0551-Load-Release
- CVE-2020-0551-Load-Prerelease
- CVE-2020-0551-CF-Release
- CVE-2020-0551-CF-Prerelease

See [Mitigation Configuration Introduction](#) for detailed information on these configurations.

---

### **NOTE:**

These four configurations are only supported on x64 platform.

---

### Mitigation Configuration Introduction

CVE-2020-0551 Mitigation includes the following four configurations.

- CVE-2020-0551-Load-Release
- CVE-2020-0551-Load-Prerelease
- CVE-2020-0551-CF-Release
- CVE-2020-0551-CF-Prerelease

CVE-2020-0551-Load-Release

- Project is built with Release mode and Load level mitigation is enabled.
- All C/C++ files will be built with `/Qspectre-load` option.

CVE-2020-0551-Load-Prerelease

- Project is built with Prerelease mode and Load level mitigation is enabled.
- All C/C++ files will be built with `/Qspectre-load` option.

CVE-2020-0551-CF-Release

- Project is built with Release mode and Control Flow level mitigation is enabled.
- All C/C++ files will be built with `/Qspectre-load-cf` option.

CVE-2020-0551-CF-Prerelease

- Project is built with Prerelease mode and Control Flow level mitigation is enabled.
- All C/C++ files will be built with `/Qspectre-load-cf` option.

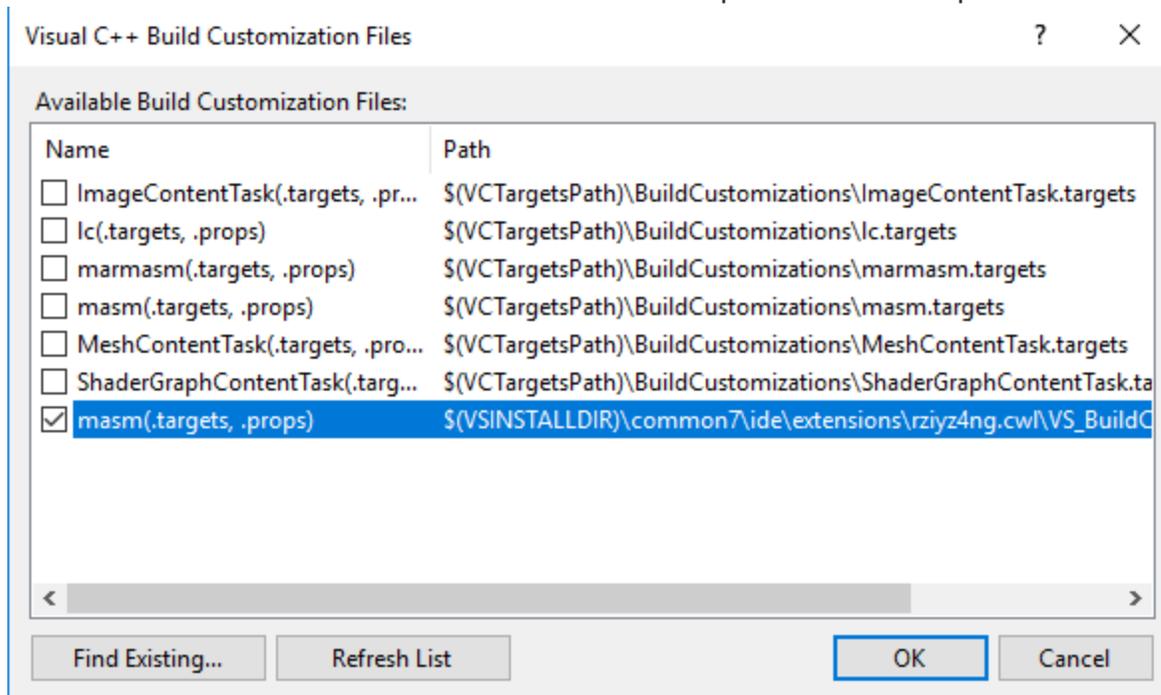
In each configuration, each .asm file's build command will be changed according to its Properties->General->Item Type type:

- If a .asm file's type is Custom Build Tool and it's built by ml64.exe. The file's Properties->Custom Build Tool->Command Line will be switched to:  

```
python "$(SGXSDKInstallPath)\scripts\sgx-asm-pp.py"
--assembler=ml64 --MITIGATION-CVE-2020-0551=
[LOAD|CF] <ml64 command line options>
```
- If a .asm file's type is Custom Build Tool and it's built by nasm.exe. The file's Properties->Custom Build Tool->Command Line will be switched to:  

```
python "$(SGXSDKInstallPath)\scripts\sgx-asm-pp.py"
--assembler=nasm --MITIGATION-CVE-2020-0551=
[LOAD|CF] <nasm command line options>
```
- If a .asm file's type is Microsoft Macro Assembler. The trusted project's Properties->Build Dependencies->Build Cus-

tomizations . . will be switched to Intel® SGX special masm script.



### SW Hardening

Affected processors will get the SW hardening needed attestation response.

### INTEL-SA-00219

On some processors, mitigation of this issue requires disabling Intel integrated graphics. As an alternative, Intel SGX developers can follow the guidance in [INTEL-SA-00219 SGX SW Guidance](#)

### SW Hardening

This issue is not associated with the SW hardening needed attestation response.

### INTEL-SA-00088 (includes Spectre v1)

One approach for mitigating Spectre v1 in SGX enclaves is described in [INTEL-SA-00088 \(Spectre v1\) SGX SW Guidance](#).

**SW Hardening**

This issue is not associated with the SW hardening needed attestation response.

## Appendix

This topic provides the following reference information:

- [Unsupported MSVC\\* Options for Enclaves](#)
- [Unsupported Intrinsics](#)
- [Unsupported C Standard Functions](#)
- [Unsupported C++ Standard Classes and Functions](#)
- [Unsupported C and C++ Keywords](#)
- [C11 Support on Windows\\* Operating System](#)
- [C++11 Support on Windows\\* Operating System](#)

### Unsupported MSVC\* Options for Enclaves

The following MSVC\* compiler options are not supported to build enclaves:

**Table 18** Unsupported MSVC Compiler Options

Option	Description	Remark
<code>/clr</code>	Enables applications and components to use features of the common language runtime.	
<code>/MD</code> <code>/MDd</code> <code>/MT</code> <code>/MTd</code>	Selects run-time library.	Linking of DLL's within enclave not allowed. Instead, use Intel® SGX trusted libraries.
<code>/EHa</code>	Exception handling model that catches both asynchronous (structured) and synchronous (C++) exceptions.	C++ exceptions are supported inside an enclave, but SEH is not supported.
<code>/fp</code>	Specify the floating-point behavior.	Not supported in the Intel® SGX version of the Intel® numeric library.
<code>/Qimprecise_await</code> <code>fwaits</code>	Removes <code>await</code> commands inside try blocks	Not supported in the Intel® SGX version of the Intel numeric library.
<code>/Qpar</code> <code>/Qpar-report</code>	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 <a href="#">assembly manifest</a> 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_labels	Causes compiler not to generate the corresponding instructions.	

### Unsupported Intrinsics

The majority of the intrinsics are valid within an enclave. The Microsoft standard intrinsic 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 intrinsic is used.

---

The following intrinsics should not be used within an enclave:

**Table 19 Unsupported MSVC Compiler Intrinsics**

<b>Not Supported: FS/GS related</b>			
<code>__addgsbyte</code>	<code>__addgsword</code>	<code>__addgsdword</code>	<code>__addgsqword</code>
<code>__incgsbyte</code>	<code>__incgsword</code>	<code>__incgsdword</code>	<code>__incgsqword</code>
<code>__writgsbyte</code>	<code>__writgsword</code>	<code>__writgsdword</code>	<code>__writgsqword</code>
<code>__addfsbyte</code>	<code>__addfsword</code>	<code>__addfsdword</code>	
<code>__incfsbyte</code>	<code>__incfsword</code>	<code>__incfsdword</code>	
<code>__writefsbyte</code>	<code>__writefsword</code>	<code>__writefsdword</code>	<code>__writefsqword</code>
<b>Not Supported: Interrupt/Debug related</b>			
<code>_enable</code>	<code>_disable</code>	<code>__halt</code>	<code>__int2c</code>
<b>Not Supported: I/O related</b>			
<code>_inp, inp</code>	<code>_inpd, inpd</code>	<code>_inpw, inpw</code>	
<code>_out, out</code>	<code>_outp, outd</code>	<code>_outw, outw</code>	
<code>__inbyte</code>	<code>__inword</code>	<code>__indword</code>	
<code>__outbyte</code>	<code>__outword</code>	<code>__outdword</code>	
<code>__inbytestring</code>	<code>__inwordstring</code>	<code>__indwordstring</code>	
<code>__outbytestring</code>	<code>__outwordstring</code>	<code>__outdwordstring</code>	
<b>Not Supported: VMX related</b>			
<code>__vmx_off</code>	<code>__vmx_on</code>	<code>__vmx_vmclear</code>	<code>__vmx_vmlaunch</code>
<code>__vmx_vmptrld</code>	<code>__vmx_vmptrst</code>	<code>__vmx_vmread</code>	<code>__svm_vmresume</code>
<code>__vmx_wmwrite</code>	<code>__nvreg_save_fence</code>	<code>__nvreg_restore_fence</code>	
<code>__svm_clgi</code>	<code>__svm_invlpga</code>	<code>__svm_skinit</code>	<code>__svm_stgi</code>
<code>__svm_vmload</code>	<code>__svm_vmrun</code>	<code>__svm_vmsave</code>	
<b>Not Supported: Architectural State related (many require ring 0 priv-</b>			

ilege)			
__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
_AddressOfReturnAd- dress	_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 an enclave; otherwise, the compilation would fail.

Table 20 Unsupported C Standard Functions

Header file	Header file in Intel® SGX?	Unsupported definition	
		Macros/Types	Functions
complex.h	Yes	complex, _complex_l, imaginary, _imaginary_l	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

			(), cproj(), creal(), crealf(), creall()
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()
signal.h	No	sig_atomic_t, SIG_DFL, SIG_ ERR, SIG_IGN, SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, SIGTERM,	signal(), raise()

<code>stdatomic.h</code>	Yes	<code>ATOMIC_&lt;TYPE&gt;_</code> <code>LOCK_FREE,</code> <code>atomic_&lt;data-</code> <code>type&gt;</code>	<code>kill_dependency(),</code> <code>atomic_fetch_and(),</code> <code>atomic_fetch_and_explicit</code> <code>(), atomic_thread_fence</code> <code>(), atomic_signal_fence()</code>
<code>stdio.h</code>	Yes	<code>FILE, fpos_t,</code> <code>_IOFBF, _</code> <code>IOLBF, _IONBF,</code> <code>FILENAME_MAX,</code> <code>FOPEN_MAX, L_</code> <code>tmpnam, SEEK_</code> <code>CUR, SEEK_END,</code> <code>SEEK_SET, TMP_</code> <code>MAX, stderr,</code> <code>stdin, stdout,</code>	<code>remove(), rename(), tmp-</code> <code>file(), tmpnam(), fclose</code> <code>(), fflush(), fopen(),</code> <code>fopen_s(), freopen(),</code> <code>freopen_s(), setbuf(),</code> <code>setvbuf(), fprintf(),</code> <code>fprintf_s(), fscanf(),</code> <code>fscanf_s(), printf(),</code> <code>scanf(), scan_s(),</code> <code>snprintf_s(), sprintf(),</code> <code>sprintf_s(), sscanf(),</code> <code>sscanf_s(), vfprintf(),</code> <code>vfprintf_s(), vfscanf(),</code> <code>vfscanf_s(), vprintf(),</code> <code>vprintf_s(), vscanf(),</code> <code>vscanf_s(), vsnprintf_s</code> <code>(), vsprintf(), vsprintf_</code> <code>s(), vsscanf(), vsscanf_s</code> <code>(), fgetc(), fgets(),</code> <code>fputc(), fputs(), getc(),</code> <code>getchar(), gets(), gets_s</code> <code>(), putc(), putchar(),</code> <code>puts(), ungetc(), fread</code> <code>(), fwrite(), fgetpos(),</code> <code>fseek(), fsetpos(), ftell</code> <code>(), rewind(), clearerr(),</code> <code>feof(), ferror(), perror</code> <code>()</code>
<code>stdlib.h</code>	Yes		<code>rand(), srand(), atexit</code> <code>(), exit(), _Exit(),</code> <code>quick_exit(), at_quick_</code> <code>exit(), getenv(), getenv_</code> <code>s(), system(), bsearch_s</code> <code>(), qsort_s(), wctomb_s</code> <code>(), wcstombs_s()</code>

string.h	Yes		strcpy(), strcat(), strstr(), strerror_s(), strerrorlen_s(), strlen_s() ( )
threads.h	No	thrd_t, thrd_ success, thrd_ timeout, thrd_ busy, thrd_ nomem, thrd_ error, thrd_ start_t, mtx_ t, mtx_ plain mtx_ recursive, mtx_ timed, cnd_t, tss_t, TSS_DTOR_ ITERATIONS, tss_dtor_t	thrd_create(), thrd_equal ( ), thrd_current(), thrd_ sleep(), thrd_yield(), thrd_exit(), thrd_detach ( ), thrd_join(), mtx_init ( ), mtx_lock(), mtx_timed- lock(), mtx_trylock(), mtx_unlock(), mxm_destroy ( ), call_once(), cnd_init ( ), cnd_signal(), cnd_ broadcast(), cnd_wait(), cnd_timedwait(), cnd_des- troy(), tss_create(), tss_get(), tss_set(), tss_delete()
time.h	Yes	timespec	clock(), mktime(), time ( ), asctime_s(), ctime(), ctime_s(), gmtime(), gmtime_s(), localtime(), localtime_s()
uchar.h	Yes	__STDC_UTF_16_ __, __STDC_UTF_ 32__	mbrtoc16() c16rtomb() mbrtoc32() c32rtomb()
wchar.h	Yes		fwprintf(), fwscanf(), swscanf(), vfwprintf(), vfwscanf(), vswscanf(), vwprintf(), vwscanf(), wprintf(), wscanf(), fgetwc(), fgetws(), fputwc(), fputws(), fwide ( ), getwc(), getwchar(), putwc(), putwchar(), ungetwc(), wcstod(),

			<code>wcstof(), wcstold(), wcstol(), wcstoll(), wcstoul(), wcstoull(), wcscopy(), wmemcpy_s(), wmemmove_s(), wscat(), wcsnlen_s(), wcsftime(), wrtomb_s(), mbsrtowcs_s() ( ), wcsrtombs_s()</code>
<code>wctype.h</code>	Yes		<code>iswalnum(), iswalpalpha(), iswblank(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(), wctype(), tolower(), toupper(), towctrans(), wctrans()</code>

(\*) The trusted standard C library does not support `char strstr(const char*, const char*)`. The trusted standard C library supports the 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 Classes and Functions

The following table lists unsupported C++11 classes and functions inside the enclave.

Table 21 Utilities library

Header File	Support	Unsupported Classes
<code>cstdlib</code>	Partial	
<code>csignal</code>	No	
<code>csetjmp</code>	No	
<code>cstdarg</code>	Yes	
<code>typeinfo</code>	Partial	
<code>typeid</code>	Yes	
<code>type_traits</code>	Yes	
<code>bitset</code>	Yes	
<code>functional</code>	Yes	
<code>utility</code>	Yes	
<code>ctime</code>	Partial	
<code>chrono</code>	No	
<code>cstdint</code>	Yes	
<code>initializer_list</code>	Yes	
<code>tuple</code>	Yes	

Table 22 Dynamic memory management

Header File	Support	Unsupported Classes
<code>memory</code>	Yes	
<code>new</code>	Partial	
<code>scoped_allocator</code>	Yes	

Table 23 Numeric limits

Header File	Support	Unsupported Classes
<code>cfloat</code>	Yes	

cinttypes	Partial	
climits	Yes	
cstdint	Partial	
limits	Yes	

Table 24 Error handling

Header File	Support	Unsupported Classes
cassert	Yes	
cerrno	Yes	
exception	Partial	nested_exception
stdexcept	Yes	
system_error	Yes	

Table 25 Strings library

Header File	Support	Unsupported Classes
cctype	Partial	
cstring	Partial	
cuchar	No	
cwchar	Partial	
cwctype	Partial	
string	Yes	

Table 26 Containers library

Header File	Support	Unsupported Classes
array	Yes	
deque	Yes	
forward_list	Yes	
list	Yes	
map	Yes	
queue	Yes	
set	Yes	
stack	Yes	
unordered_map	Yes	
unordered_set	Yes	
vector	Yes	

Table 27 Algorithms library

Header File	Support	Unsupported Classes
algorithm	Partial	

Table 28 Iterators library

Header File	Support	Unsupported Classes
iterator	Yes	

Table 29 Numerics library

Header File	Support	Unsupported Classes
cfenv	No	
cmath	Partial	
complex	Yes	
numeric	Yes	
random	No	
ratio	Yes	
valarray	Yes	

Table 30 Input/Output library

Header File	Support	Unsupported Classes
cstdio	Partial	
fstream	No	
iomanip	No	
ios	No	
iosfwd	No	
iostream	No	
istream	No	
ostream	No	
sstream	No	
streambuf	No	
stringstream	No	

Table 31 Localization library

Header File	Support	Unsupported Classes
locale	No	
codecvt	No	
locale	No	

Table 32 Regular expressions library

Header File	Support	Unsupported Classes
regex	No	

Table 33 Atomic operations library

Header File	Support	Unsupported Classes
atomic	Yes	

Table 34 Thread library

Header File	Support	Unsupported Classes
condition_variable	Partial	
future	No	
mutex	Partial	timed_mutex, recursive_timed_mutex
thread	No	

Table 35 C compatibility headers

Header File	Support	Unsupported Classes
ccomplex	Yes	
ciso646	Yes	This header file is empty in a conforming implementation
cstdalign	Yes	
cstdbool	Yes	
ctgmath	Yes	

### Unsupported C and C++ Keywords

The following keywords are not supported in an enclave:

Table 36 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	<code>__identifier</code> , <code>value class</code> , <code>delegate</code> , <code>enum class*</code> , <code>typeid</code> , <code>enum struct*</code> , <code>generic</code> , <code>event</code> , <code>finally</code> , <code>initonly</code> , <code>'for each, in'</code> , <code>ref struct</code> , <code>friend_as</code> , <code>gcnew</code> , <code>safecast</code> , <code>interface class</code> , <code>interface struct</code> , <code>ref class</code> , <code>interior_ptr</code> , <code>literal</code> , <code>value struct</code> , <code>property</code>
Timing and Event	<code>__event</code> , <code>__hook</code> , <code>__unhook</code> , <code>__raise</code> , <code>event</code>
Additional Keywords	<code>dllimport</code> , <code>__unaligned</code> , <code>__w64</code>

(\*) The Visual Studio extensions `enum class` and `enum struct` are not supported, see <https://msdn.microsoft.com/en-us/library/a6cskb49.aspx>. However, the C++11 `enum class` and `enum struct` features are supported.

### C11 Support on Windows\* Operating System

Although C11 is considered the baseline, the availability of certain C11 features depend on the Visual Studio\* and compiler version and/or a specific compiler option.

C11 Language Feature	Supported
Anonymous struct and union	Yes
Atomic operations	Yes
Macros for complex values	Yes*
New macros in float.h	Yes
<code>static_assert</code>	Yes
Typedef redefinition	Yes
Unicode strings	Yes
<code>uchar.h</code>	Partially
<code>_Alignas</code> , <code>_Alignof</code> , <code>max_align_t</code> , <code>stdalign.h</code>	Yes*
<code>_Atomic</code>	No
<code>_Generic</code>	Yes*
<code>_Noreturn</code> , <code>stdnoreturn.h</code>	Yes*
<code>__STDC_VERSION__ == 201112L</code>	Yes*
<code>_Static_assert</code>	Yes*
<code>_Thread_local</code>	Yes*
<code>aligned_alloc</code>	Yes

## C++11 Support on Windows\* Operating System

Although C++11 is considered the baseline, the availability of certain C++11 features depend on the Visual Studio\* and compiler version and/or a specific compiler option.

<b>C++11 Language Feature</b>
Additional type traits*
Alias templates
Alignment (alignas, alignof)
Allow typename outside of templates
Attributes
Bidirectional fences*
C99 preprocessor
Conversions of lambdas to function pointers
Data-dependency ordering*
Data-dependency ordering: function annotation
Default template arguments for function templates
Defaulted and deleted functions
Delegating constructors
Explicit conversion operators
Expression SFINAE
Extended friend declarations
Extended sizeof
Extern templates
Forward declared enums
Inheriting constructors
Initialization of class objects by rvalues
Initializer lists*
Inline namespaces
Lambdas
Local and unnamed types as template arguments
Magic statics
Memory model*
Minimal support for garbage collection*
Non-static data member initializers
R-value references
Range-based for-loop
Raw string literals

<b>C++11 Language Feature</b>
Reworded sequence points
Right angle brackets
Standard-layout and trivial types*
Strong compare and exchange*
Strongly typed enums
Thread-local storage (thread_local)*
Type traits*
Unicode strings literals
Universal character names in literals
Unrestricted unions
User-defined literals
Variadic templates
__func__
auto
char16_t/char32_t*
constexpr**
decltype
long long
nullptr
noexcept
override and final
static_assert

(\*) You do not need to use the compiler option /Qstd=c++11 to access any of these features if you develop with the Visual Studio\*.

### Supported C Secure Functions

The following table lists supported C secure functions inside the enclave.

Table 1 Supported C Secure Functions

Header file	Supported C Secure Functions
mbusafecrt.h	strcat_s(), wscat_s(), strncat_s(), wcsncat_s(), strcpy_s(), wcsncpy_s(), strncpy_s(), wcsncpy_s(), strtok_s(), wcstok_s(), wcsnlen(), _itoa_s(), _itow_s(), _ltoa_s(), _ltow_s(), _ultoa_s(), _ultow_s(), _i64toa_s(), _i64tow_s(), _ui64toa_s(), _ui64tow_s(), sprintf_s(), swprintf_s(), _snprintf_s(), _snwprintf_s(), _vsprintf_s(), _vsnprintf_s(), _vswprintf_s(), _vsnwprintf_s(), memcpy_s(), memmove_s()