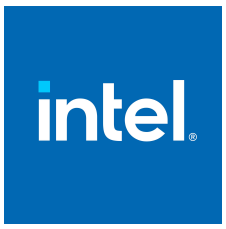

Intel® Software Guard Extensions (Intel® SGX) SDK for Linux* OS

Developer Reference

Legal Information

Revision History

Revision Number	Description	Revision Date
1.5	Intel® SGX Linux 1.5 release	May 2016
1.6	Intel® SGX Linux 1.6 release	September 2016
1.7	Intel® SGX Linux 1.7 release	December 2016
1.8	Intel® SGX Linux 1.8 release	March 2017
1.9	Intel® SGX Linux 1.9 release	July 2017
2.0	Intel® SGX Linux 2.0 release	November 2017
2.1	Intel® SGX Linux 2.1 release	December 2017
2.1.1	Intel® SGX Linux 2.1.1 release	March 2018
2.1.2	Intel® SGX Linux 2.1.2 release	March 2018
2.1.3	Intel® SGX Linux 2.1.3 release	April 2018
2.2	Intel® SGX Linux 2.2 release	July 2018
2.3	Intel® SGX Linux 2.3 release	September 2018
2.4	Intel® SGX Linux 2.4 release	November 2018
2.5	Intel® SGX Linux 2.5 release	March 2019
2.6	Intel® SGX Linux 2.6 release	June 2019
2.7	Intel® SGX Linux 2.7 release	September 2019
2.7.1	Intel® SGX Linux 2.7.1 release	November 2019
2.8	Intel® SGX Linux 2.8 release	January 2020
2.9	Intel® SGX Linux 2.9 release	March 2020
2.9.1	Intel® SGX Linux 2.9.1 release	April 2020
2.10	Intel® SGX Linux 2.10 release	June 2020
2.11	Intel® SGX Linux 2.11 release	August 2020
2.12	Intel® SGX Linux 2.12 release	November 2020
2.13	Intel® SGX Linux 2.13 release	January 2021
2.14	Intel® SGX Linux 2.14 release	June 2021
2.15	Intel® SGX Linux 2.15 release	September 2021
2.15.1	Intel® SGX Linux 2.15.1 release	November

		2021
2.16	Intel® SGX Linux 2.16 release	March 2022
2.17	Intel® SGX Linux 2.17 release	June 2022

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 shared object on Linux* OS.

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. Linux*, however, can increase the protected memory size through paging.

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.[1]

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

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

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

	enclave (QE), provisioning enclave (PvE), launch enclave (LE)).
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.
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 using the features of Intel® Software Guard Extensions SDK to create and manage Intel® SGX application projects.

Creating Intel® Software Guard Extensions Projects

To create an Intel® Software Guard Extensions project on Linux* OS, you should follow the directory structure and Makefiles from one of the sample applications in the Intel® SGX SDK. In an Intel SGX project, you should normally prepare the following files:

1. Enclave Definition Language (EDL) file - describes enclave trusted and untrusted functions and types used in the function prototype. See Enclave Definition Language Syntax for details.
2. Enclave Configuration File (ECF) - contains the information of the enclave metadata. See Enclave Configuration File for details.
3. Signing key files - used to sign an enclave to produce a signature structure that contains enclave properties such as enclave measurement. See Signing Key Files for details.
4. Application and enclave source code - the implementation of application and enclave functions.
5. makefile - it performs the following steps:
    1. Generates edger routines (see Edger8r Tool for details).
    2. Builds the application and enclave.
    3. Signs the enclave (see Enclave Signing Tool for details).
6. Linker script file - you should use the linker script to hide all unnecessary symbols, and only export enclave_entry, g_global_data, and g_global_data_sim.

Once you understand how an Intel SGX application is built, you may customize the project setup according to your needs.

To develop an Intel SGX application, Intel® SGX SDK supports a few non-standard configurations, not present in other SDKs. Enclave Project Configurations explains the various enclave project configurations as well as the corresponding Makefile options.

Enclave Image Generation

An enclave image is a statically linked shared object under Linux OS, without any external dependencies. You must follow the guidelines below to generate a proper enclave image:

1. Link the tRTS with the `--whole-archive` option, so that the whole content of the trusted runtime library is included in the enclave;

2. For other libraries, you just need to pull the required symbols. For example, if an enclave requires routines in the trusted standard C and seal libraries:

HW mode:

```
--start-group -lsgx_tstdc -lsgx_tservice -lsgx_tcrypto --end-group
```

Simulation mode:

```
--start-group -lsgx_tstdc -lsgx_tservice_sim -lsgx_tcrypto --end-group
```

In addition, a linker script is also recommended to hide all unnecessary symbols.

```
// file: enclave.lds

enclave.so

{
    global:
    enclave_entry;
    g_global_data_sim;
    g_peak_heap_used;
    g_peak_rsrv_mem_committed;
    local:
    *;
};
```

The symbol `enclave_entry` is the entry point to the enclave. The symbol `g_global_data_sim` comes from the tRTS simulation library and is required to be exposed for running an enclave in the simulation mode since it distinguishes between enclaves built to run on the simulator and on the hardware. The `sgx_emmt` tool relies on the symbol `g_peak_heap_used` to determine the size of the heap that the enclave uses and relies on the symbol `g_peak_rsrv_mem_committed` to determine the size of the reserved memory that the enclave uses. The symbol `__ImageBase` is used by tRTS to compute the base address of the enclave.

Assuming that you have a few object files to be linked into a target enclave, use the command line similar to the following:

```
$ ld -o enclave.so file1.o file2.o \
-pie -eenclave_entry -nostdlib -nodefaultlibs -nostart-files --no-undefined \
--whole-archive -lsgx_trts --no-whole-archive \
--start-group -lsgx_tstdc --lsgx_tservice -lsgx_crypto --end-group \
-Bstatic -Bsymbolic --defsym=__ImageBase=0 --export-dynamic \
--version-script=enclave.lds
```

You are also encouraged to help harden your enclaves, by passing one of the following options to the linker, to put read-only non-executable sections in your own segment:

```
ld.gold --rosegment
```

or,

```
-Wl,-fuse-ld=gold -Wl,--rosegment
```

Using Intel® Software Guard Extensions Eclipse* Plug-in

The Intel® Software Guard Extensions Eclipse* Plug-in helps the enclave developer to maintain enclaves and untrusted related code inside Eclipse* C/C++ projects.

To get more information on Intel® Software Guard Extensions Eclipse* Plug-in, see Intel® Software Guard Extensions Eclipse* Plug-in Developer Guide from the Eclipse Help content: **Help > Help Contents > Intel® SGX Eclipse Plug-in Developer Guide**.

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
`--use-prefix`	Prefix the untrusted proxy with the enclave name.
`--header-only`	Generate header files only.
`--search-path <path>`	Specify the search path of EDL files.
`--untrusted`	Generate untrusted proxy and bridge routines only.
`--trusted`	Generate trusted proxy and bridge routines only.
`--untrusted-dir <dir>`	Specify the directory for saving the untrusted code.
`--trusted-dir <dir>`	Specify the directory for saving the trusted code.
`--help`	Print 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 1 Enclave Signature Structure

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

Section	Name
Body	MISCSELECT
	MISCMASK
	RESERVED
	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 2 Signing Tool Commands

Command	Description	Arguments
`sign`	Sign the enclave using the private key in one step.	Required: -enclave, -key, -out Optional: -config, -dumpfile, -cssfile
`gendata`	The first step of the 2-step signing process. Generate the enclave signing material to be signed by an external tool. This step dumps the signing material, which consists of the header and body sections of the enclave 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 `gendata` command. At this step, the signature and buffer sections are generated. The signature and buffer sections together with the header and body sections complete the enclave signature structure (see the Table Enclave Signature Structure in this topic).	Required: -enclave, -key, -out, -sig, -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 `-dumpfile` option and the SIGSTRUCT to the file specified by the `-cssfile` option.	Required: -enclave, -dumpfile Optional: -cssfile

All the valid command options are listed below:

Table 3 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 `dump` command and an optional argument for the other three commands.	
`-out <file>`	Specify the output file. It is required for the following three commands.	
	Command	**Description**
	`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 File Formats 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.	
`-ignore-rel-error`	By default, `sgx_sign` provides an error for enclaves with text relocations. You can ignore the error and continue signing by providing this option. But it is recommended that you eliminate the text relocations instead of bypassing the error with this option.	
`-ignore-init-sec-error`	By default, `sgx_sign` provides an error for enclaves with .init section. You can ignore the error and continue signing by providing this option. But it is recommended you eliminate the section instead of bypassing the error with this option.	
`-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.

`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 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 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 runs the signing tool with the `catsig`command providing the necessary information at the command line 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 4 Signing Tool File Formats

File	Format	Description
Enclave file	Shared Object	A standard Shared Object.
Signed enclave file	Shared Object	`sgx_sign` generates the signed enclave file , which includes the signature, to the enclave file.
Configuration file	XML	See Enclave Configuration File.
Key file	PEM	Key file should follow the PEM format which contains an unencrypted RSA 3072-bit key. The public exponent must be 3.
Enclave hex file	RAW	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.so -config config.xml
  -out enclave_signed.so -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.so -config con-
     fig.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.so -config con-
     fig.xml -out enclave_signed.so -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.so -dumpfile
metadata_info.txt
```

A single enclave signing tool is provided, which allows for operating with 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

You can leverage the helper script `sgx-gdb` to debug your enclave applications. To debug an enclave on a hardware platform, set the `<DisableDebug>` configuration parameter should to 0 in the enclave configuration file `config.xml`, and the `Debug` parameter to 1 in the `sgx_create_enclave(...)` that creates the enclave. Debugging an enclave is similar to debugging a shared library. However, not all standard features are available to debug enclaves. The following table lists severalunsupported GDB commands. `sgx-gdb` also supports measuring the enclave stack/heap usage by the Enclave Memory Measurement Tool. See the Enclave Memory Measurement Tool for additional information. Note that `sgx-gdb` does not support debugging enclaves created from memory by `sgx_create_enclave_from_buffer_ex` API.

Table 5 GDB Unsupported Commands

GDB Command	Description
info sharedlibrary	Does not show the status of the loaded enclave.
next/step	Does not allows to execute the next/step outside the enclave from inside the enclave. To go outside the enclave use the finish command.
call/print	Does not support calling outside the enclave from within an enclave function, or calling inside the enclave from a function in the untrusted

	domain.
charset	Only supports the GDB default charset.
gcore	Does not support debug enclave with the application dump file

Performance Measurement using Intel® VTune(TM) Amplifier

You can use the Intel® VTune™ Amplifier Application 2016 Update 2 and higher to measure the performance of the Intel® Software Guard Extensions (Intel® SGX) applications, including the enclave. The Intel® VTune Amplifier application supports a new analysis type called `SGX Hotspots` that can be used to profile Intel® SGX enclave applications. You can use the default settings of the `SGX Hotspots` to profile the application and the enclave code. The precise event based sampling (PEBS) helps profiling Intel® SGX enclave code. You can add precise events (`_PS` events) to the collection to profile enclave code. Non-precise events cannot help profiling Intel® SGX enclave code.

You can use the Intel® VTune™ Amplifier to measure the performance of the enclave code only when the enclave is launched as a debug enclave. To launch as a debug enclave, pass a value of `1` as the second parameter to the `sgx_create_enclave` function, which loads and initializes the enclave as shown below. Use the pre-defined macro `SGX_DEBUG_FLAG` as the parameter which is automatically set to 1 in `DEBUG` and `PRE-RELEASE` modes.

```
sgx_create_enclave(ENCLAVE_FILENAME, SGX_DEBUG_FLAG,
&token, &updated, &global_eid, NULL);
```

NOTE:

Only use the Intel® VTune™ Amplifier to measure the performance in the `DEBUG` and `PRE-RELEASE` modes because `SGX_DEBUG FLAG` cannot be `1` when launching an enclave in the `RELEASE` configuration.

The Intel® VTune™ Amplifier provides two options to profile applications:

- Run the applications using the Intel® VTune™ Amplifier. If you use this approach, no additional steps required.
- Attach to the running process or the enclave application. If you use this approach, define the environment variable as follows:
    - For 64bit:

      ```
      INTEL_LIBITTNOTIFY64 = <VTune Installation
      Dir>/lib64/runtime/ittnotify_collector.so
      ```

Once the enclave is loaded, the invoked instrumentation and tracing technology (ITT) API of the Intel® VTune™ Amplifier in the uRTS passes information about the enclave to the Intel® VTune Amplifier and profiles the Intel® SGX enclave applications. When you attach the Intel VTune Amplifier to the application after invoking the ITT API of the Intel® VTune Amplifier, the module information about the enclave is cached in the ITT dynamic library and is used by the Intel® VTune Amplifier to attach to the process. The following table describes different scenarios of how the Intel VTune Amplifier is used to profile the enclave application:

Intel® VTune(TM) Amplifier Invocation	**Additional Configuration**	**ITT API Result**	**uRTS Action**
Launch the application with the Intel® VTune (TM) Amplifier	N/A	Intel® VTune(TM) Amplifier is profiling	Set the Debug OPTIN bit and invoke the Module mapping API.
Late attach *before* invoking the ITT API for the Intel® VTune(TM) Amplifier profiling check in `sgx_cereate_enclave`	ITT environment variable is set.	Intel® VTune(TM) Amplifier is profiling	Set the Debug OPTIN bit and invoke the Module mapping API.
	ITT environment variable is not set.	Intel® VTune(TM) Amplifier is not profiling	Do not set the Debug OPTIN bit and do not invoke the Module mapping API. Even though the Intel® VTune(TM) Amplifier is running, it cannot profile enclaves. You need to set the environment variable.
Late attach *after* invoking theITT API for the Intel® VTune(TM) Amplifier profiling check `in sgx_cereate_enclave`	ITT environment variable is set.	Intel® VTune(TM) Amplifier is profiling	Set the Debug OPTIN bit and invoke the Module mapping API. The ITT dynamic library caches the module information and provides it to the

			Intel® VTune(TM) Amplifier during attaching to the process.
	ITT environment variable is not set.	Intel® VTune(TM) Amplifier is not profiling	Do not set the Debug OPTIN bit and do not invoke the Module mapping API. Even though the Intel® VTune(TM) Amplifier is running, it cannot profile enclaves. You need to set the environment variable.
Launch the application without the Intel® VTune (TM) Amplifier	N/A	Intel® VTune(TM) Amplifier is not profiling	Do not set the Debug OPTIN bit and do not invoke the Module mapping API.

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.

On Linux* OS, the enclave memory measurement capability is provided by the helper script `sgx-gdb`. The `sgx-gdb` is a GDB extension for you to debug your enclave applications. See Enclave Debugger for details.

To measure how much protected memory an enclave uses, you should leverage `sgx-gdb` to launch GDB with `sgx_emmt` enabled and load the test

application that is using the enclave. You may also attach the debugger to a running Intel SGX application to measure the memory usage of the enclave.

The `sgx-gdb` provides three commands pertaining the `sgx_emmt` tool:

Table 6 Enclave Memory Measurement Tool Commands

Command	Description
enable sgx_emmt	Enable the enclave memory measurement tool.
disable sgx_emmt	Disable the enclave memory measurement tool.
show sgx_emmt	Show whether the enclave memory measurement tool is enabled or not.

Here are the typical steps necessary to collect an enclave's memory usage information:

1. Leverage `sgx-gdb` to start a GDB session.

2. Enable the enclave memory measurement function with `enable sgx_emmt`.

3. Load and run the test application which is using the enclave.

NOTE:
To collect peak stack/heap/reserved memory usage for an enclave on a hardware platform correctly, make sure the enclave meets the following requirements:

1. The enclave is debuggable. This means that the `<DisableDebug>` configuration parameter in the enclave configuration file should be set to `0`.
2. The enclave is launched in the debug mode. To launch the enclave in the debug mode, set the debug flag to `1` when calling sgx_create_enclave to load the enclave.
3. Export `g_peak_heap_used` and g_peak_rsrv_mem_committed in the version script of the enclave.
4. Destroy the enclave by using thesgx_destroy_enclave API.

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.

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 7 CPUSVN Configuration Tool Commands

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

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. Define the interface between the untrusted application and the enclave in the EDL file.
2. Implement the application and enclave functions.
3. 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.
4. Run and debug the application in simulation and hardware modes. See Enclave Debugger for more details.
5. 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).Linux* Shared Objects 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 8 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 Shared Objects	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	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. Spin-locks, trusted mutex and condition variables API can be used for thread synchronization inside the enclave.
Thread Local Storage (TLS)	Partially	Only implicitly via __thread.
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 Custom Exception Handling for more details.
Signals	No	Signals are not supported inside an enclave.

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 Linux* OS loads shared objects. 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, reader/writer lock, condition variable, and CPUID APIs from libsgx_tstdc.a make OCALLs. Similarly, the trusted support

library libsgx_tservice.a, 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 `libsgx_tstdc.a`.

```
libsgx_tstdc.a(sethread_mutex.o): In function `sgx_
thread_mutex_lock':

sethread_mutex.cpp:109: undefined reference to `sgx_
thread_wait_untrusted_event_ocall'

libsgx_tstdc.a(sethread_mutex.o): In function `sgx_
thread_mutex_unlock':

sethread_mutex.cpp:213: undefined reference to `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 `.a` file containing the trusted code and data, may also include an `.edl` file as well as an untrusted `.a` 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 Eclipse plug-in. 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 SO file.

Enclaves, like regular SO 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 SO 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 shared object 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.

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 9 Trusted Libraries included in the Intel® SGX SDK

Name	Description	Comment
`libsgx_trts.a`	Intel® SGX Runtime library	Must link when running in HW mode
`libsgx_trts_sim.a`	Intel® SGX Runtime library (simulation mode)	Must link when running in simulation mode
`libsgx_tstdc.a`	Standard C library (math, string, and so on.)	Must link
`libsgx_tcxx.a,`	Standard C++ libraries, STL	Optional
`libsgx_tservice.a`	Data seal/unseal (encryption), trusted Architectural Enclaves support, Elliptic Curve Diffie-Hellman (EC DH) library, and so on.	Must link when using HW mode
`libsgx_tservice_sim.a`	The counterpart of libsgx_tservice.a for simulation mode	Must link when using simulation mode
`libsgx_tcrypto.a`	Cryptographic library	Must link
`libsgx_tkey_exchange.a`	Trusted key exchange library	Optional
`libsgx_tprotected.a`	Protected File System library	Optional
`libsgx_tswitchless.a`	Switchless Enclave Function Calls	Optional
`libsgx_pcl.a`	Enables Intel® SGX Protected Code Loader for enclave code confidentiality	Optional
`libsgx_pclsim.a`	Enables Intel® SGX Protected Code Loader for enclave code confidentiality (simulation mode)	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 10 Untrusted Libraries included in the Intel® SGX SDK

Name	Description	Comment
`libsgx_urts.so`	Provides functionality for applications to manage enclaves	Must link when running in HW mode. `libsgx_urts.so` is included in Intel® SGX PSW
`libsgx_urts_sim.so`	uRTS library used in simulation mode	Dynamically linked
	Provides both enclaves and untrusted applications access to services provided by the AEs	Must link when running in HW mode. is included in Intel® SGX PSW
`libsgx_uae_service_sim.so`	Untrusted AE support library used in simulation mode	Dynamically linked
`libsgx_ukey_exchange.a`	Untrusted key exchange library	Optional
`libsgx_uae_service.so`	Provides both enclaves and untrusted applications access to services	Must link when running in HW mode. This library is being phased out. We recommend start linking

	provided by the AEs	with the libraries below and corresponding header files based on the services your application needs.
libsgx_launch.so	Provides applications access to launch services provided by the AEs.	Must link when running in HW mode.
libsgx_epid.so	Provides applications access to the EPID provisioning services provided by the AEs.	Must link when running in HW mode.
libsgx_platform.so	Provides applications access to platform services (trusted time and monotonic counter).	Must link when running in HW mode.
libsgx_quote_ex.so		Must link when running in HW mode.

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 11 EDL Reserved Keywords

Data Types					
char	short	int	float	double	void
int8_t	int16_t	int32_t	int64_t	size_t	wchar_t
uint8_t	uint16_t	uint32_t	uint64_t	unsigned	struct
union	enum	long			
Pointer Parameter Handling					
in	out	user_check	count	size	readonly
isptr	string	wstring			
Others					
enclave	from	import	trusted	untrusted	include
public	allow	isary	const	propagate_errno	transition_using_threads

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 12 Behavior of wrapper functions when using the in/out attributes

	ECALL	**OCALL**
user_check	Pointer is not checked. Users must perform the check and/or copy.	Pointer is not checked. Users must perform the check and/or copy
in	Buffer copied from the application into the enclave. Afterwards, changes will only affect the buffer inside enclave.	Buffer copied from the enclave to the application. Must be used if pointer points

	Safe but slow.	to enclave data.
out	Trusted wrapper function will allocate a buffer to be used by the enclave. Upon return, this buffer will be copied to the original buffer.	The untrusted buffer will be copied into the enclave by the trusted wrapper function. Safe but slow.
in, out	Combines `in` and `out` behavior. Data is copied back and forth.	Same as ECALLs.

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 `wsting` 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 musted 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_foot_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 ephemerally. 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 `pBuff2ptr` 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` uses `gcc` 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.

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 {

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

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

        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 `replace_secret`
`clear_secret`	Y	Y
`get_secret`	Y	N
`set_secret`	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>
    <StackMinSize>0x2000</StackMinSize>
    <HeapMaxSize>0x100000</HeapMaxSize>
    <HeapMinSize>0x40000</HeapMinSize>
    <HeapInitSize>0x80000</HeapInitSize>
    <TCSNum>3</TCSNum>
    <TCSMaxNum>4</TCSMaxNum>
    <TCSMinPool>2</TCSMinPool>
    <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 be used.

Table 13 Enclave Configuration Default Values

Tag	Description	Default Value
`ProdID`	ISV assigned Product ID.	0
`ISVSVN`	ISV assigned SVN.	0
`TCSNum`	The number of TCS. Must be greater than 0.	1
`TCSMaxNum`	The maximum number of TCS. Must be greater than 0.	1
`TCSMinPool`	The minimum number of available TCS at any time in the life cycle of an enclave	1
`TCSPolicy`	TCS management policy. 0 – TCS is bound to the untrusted thread. 1 – TCS is not bound to the untrusted thread.	1
`StackMinSize`	The minimum stack size per thread. Must be 4KB aligned.	0x2000
`StackMaxSize`	The maximum stack size per thread. Must be 4KB aligned.	0x40000
`HeapInitSize`	The initial heap size for the process. Must be 4KB aligned.	0x1000000
`HeapMinSize`	The minimum heap size for the process. Must be 4KB aligned.	0x1000
`HeapMaxSize`	The maximum heap size for the process. Must be 4KB aligned.	0x1000000
`ReservedMemMaxSize`	The maximum reserved memory size for the process. Must be 4KB aligned.	0x0000000
`ReservedMemMinSize`	The minimum reserved memory size for the process. Must be	0x0000000

	4KB aligned.	
`ReservedMemInitSize`	The initial reserved memory size for the process. Must be 4KB aligned.	0x0000000
`ReservedMemExecutable`	The reserved memory is executable. **Note:** This value is only used for the Intel® SGX 1 platform.	0 - Reserved memory is not executable 1 - Reserved memory is executable
`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 8 bytes)	0
`EnclaveImageAddress`	The base address of the enclave image file.	0
`ELRangeStartAddress`	The base address of the enclave address range.	0
`PKRU`	Enable the Protection Keys	0

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

`TCSMaxNum`, `TCSNum`, and `TCSMinPool` are used to determine how many threads will be created after the enclave initialization, and how many threads

can be created dynamically when the enclave is running. These two kinds of threads are referred to as static threads and dynamic threads respectively.

`StackMaxSize` and `StackMinSize` have different meanings to a static thread and a dynamic thread.

For a static thread, only `StackMaxSize` is relevant, which determines the maximum amount of stack available.

For a dynamic thread, `StackMinSize` is the amount of stack available once the thread is created and initialized. The gap between `StackMinSize` and `StackMaxSize` is the amount of stack that is not available currently but can be expanded as necessary later. Therefore, `StackMaxSize` is the total amount of stack a thread can use. `StackMinSize` can be regarded as the lower limit and `StackMaxSize` is the upper limit.

When an enclave created with the Linux* 2.0 SDK is executing on an Intel® SGX 2.0 platform that is running the Intel® SGX 2.0 PSW, `HeapMinSize` is the amount of heap available once the enclave completes initialization.

`HeapMaxSize` is the total amount of heap an enclave can use. The gap between `HeapMinSize` and `HeapMaxSize` is the amount of heap that is not available currently but can be expanded as necessary later.

When an enclave created with the Linux* 2.0 SDK is executing on an older Intel SGX platform or a platform running a previous version of PSW, the values are interpreted differently. In this case `HeapInitSize` is the only relevant field and it indicates the total amount of heap available to an enclave.

`ReservedMemMinSize`, `ReservedMemMaxSize` and `ReservedMemInitSize` can be used to configure a reserved memory area for an enclave. By default, an enclave has no reserved memory. You can add a reserved memory area by specifying these fields in the enclave's configuration file. The reserved memory area will be added at the end of the enclave at the loading time and can be used at the runtime.

`ReservedMemExecutable` can be used to configure whether the reserved memory has executable permission by setting `ReservedMemExecutable` to `1`.

NOTE:

On the Intel® SGX 2.0 platform, the reserved memory is forcely configured to RW permission although `ReservedMemExecutable` is set to `1`.

Currently, the reserved memory is used to support Just in Time (JIT) usage in the Intel® SGX DNNL Library.

Several sample configuration files are provided in SampleEnclave project in order to further clarify the effects of different combinations of the parameter settings.

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.

The total amount of stack and heap actually used by an enclave can be measured by using the measurement tool `sgx_emmt`. See Enclave Memory Measurement Tool for details.

An Eclipse* plug-in named **Intel® SGX Update Configuration** is provided to help you easily edit your configuration file. See the *Intel® SGX Eclipse* Plug-in User's Guide* from the Eclipse's Help content 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.

`EnclaveImageAddress`, `ElRangeStartAddress` and `ElRangeSize` can be used to configure enclave elrange for an enclave. By default, these fields are set to 0. You can specify these fields in the enclave's configuration file.

`PKRU` can be used to enable or disable the Protection Keys inside enclave. Below lists the acceptable value and the corresponding meaning for this field:

- 0 - The feature must be disabled
- 1 - The feature must be enabled
- 2 - Let the loader choose to enable the feature or not.

Enclave Project Configurations

Depending on the development stage you are at, choose one of the following project configurations to build an enclave:

- Simulation: Under the *simulation* mode the enclave can be either built with debug or release compiler settings. However, in both cases the

enclave is launched in the *enclave debug* mode. The Eclipse* plugin provides the **Intel® SGX Simulation** and **Intel® SGX Simulation Debug** configuration options to enable compiling and launching the enclave in the simulation mode.From the command line, an enclave can be built in this mode by passing `SGX_DEBUG=1` for debug simulation and no parameters for release simulation. This is the default build mode. Single-step signing is the default method to sign a simulation enclave.

- Debug: When the **Intel® SGX Hardware Debug**configuration option is selected for an enclave project in Eclipse* plugin, the enclave is compiled in the *debug* mode and the resulting enclave file will contain debug information and symbols.To use this configuration for an enclave, set `SGX_MODE=HW` and `SGX_DEBUG=1` as parameters to the Makefile during the build. 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 **Intel® SGX Hardware Prerelease** configuration option for an enclave project, Eclipse* plugin will build the enclave in *release* mode with compiler optimizations applied.An enclave is built in this mode by setting `SGX_MODE=HW` and `SGX_PRERELEASE=1` in the Makefile during build. Under this configuration, the enclave is launched in *enclave debug* mode. The Makefile of the sample application defines the `EDEBUG` flag when `SGX_PRERELEASE=1` is passed as a command line parameter to the Makefile during build. 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 **Intel® SGX Hardware Release** configuration option for an Eclipse plugin 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`.This mode is enabled in enclave by passing `SGX_MODE=HW` to the Makefile while building the project. `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 shared object. To use an enclave, the enclave.so should be loaded into protected memory by calling the API sgx_create_enclave() or sgx_create_encalve_ex(). The enclave.so must be signed by sgx_sign. Before the Intel® SGX 2.4 release, 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. Starting the Intel® SGX 2.4 release, you do not need to pass and store the launch token anymore. 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.so")


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.

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 . In addition, the enclave must be linked with the `libsgx_tswitchless.a` library (see Switchless Calls Library). If the application creates an enclave that supports Switchless Calls using `sgx_create_enclave`, the enclave is created, but the Switchless Calls mode of operation is disabled and all calls are using enclave switches. If **transition_using_threads** attribute is used, but the enclave is not linked with `libsgx_twitchless.a` library, creating the enclave using `sgx_create_enclave_ex` will return SGX_ERROR_UNEXPECTED.

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

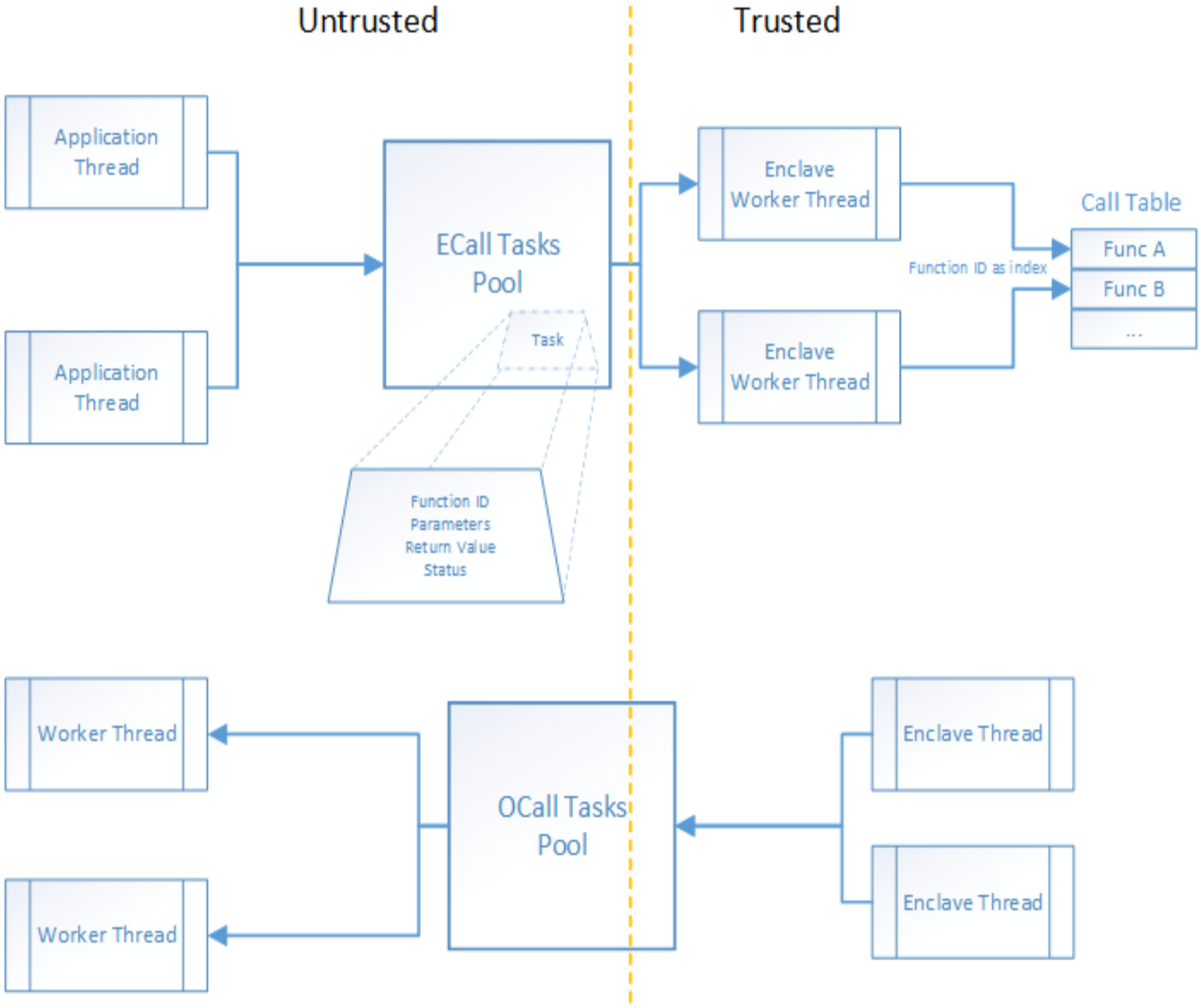

Figure 1 Switchless Calls Architecture

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
uint64_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 Linux* 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 shared object.

Solution: Encrypt the enclave shared object (.so) 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.

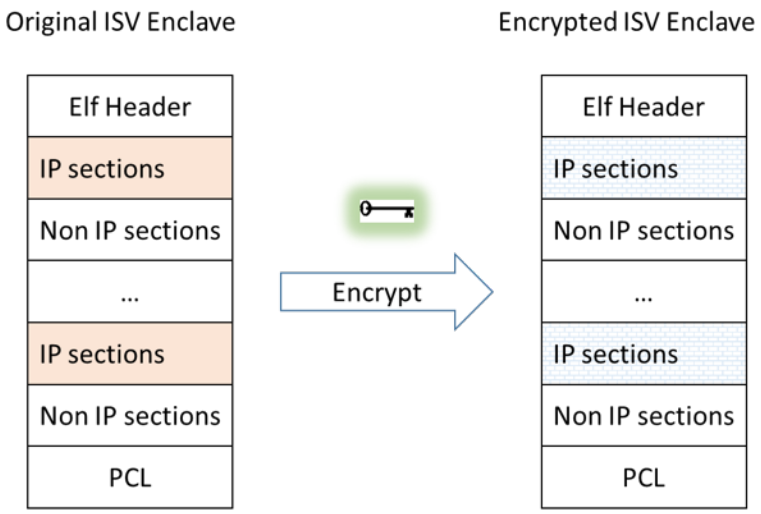

Figure 2 **Intel® SGX PCL Build Flow**

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 shared object is modified so that ELF 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 'Sections 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:

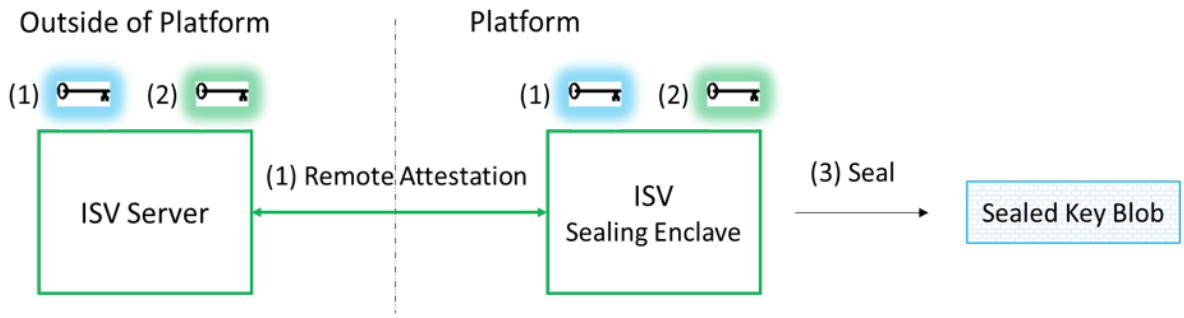

Figure 3 ISV Sealing Enclave Flow

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:

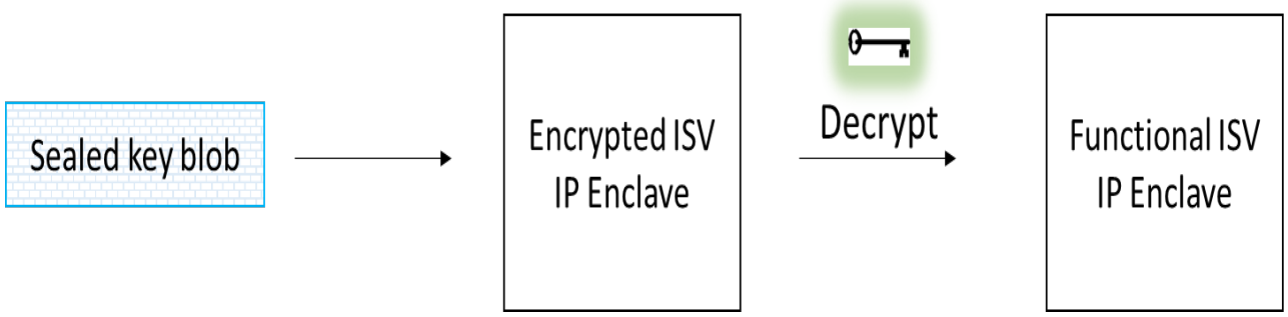

Figure 4 ISV IP Enclave Loading Flow

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	• **Link**: ISV archives and objs are linked to Enclave.so • **Sign**: Enclave.so is signed to generate Enclave.signed.so	• **Link**: ISV archives, objs, and libsgx_pcl.a are linked to IPEnclave.so • Encrypt IPEnclave.so to IPEnclave.so.enc • **Sign**: IPEnclave.so.enc is signed to generate IPEnclave.signed.so
Enclave Load	1. Enclave application	1. Enclave application gets the sealed

	loads the enclave using sgx_create_enclave. 2. sgx_create_enclave performs an implicit ecall. 3. The implicit ecall initiates an enclave runtime initialization flow.	decryption key. 2. Enclave application loads the enclave using sgx_create_enclave_ex, providing the sealed decryption key 3. sgx_create_enclave_ex performs an implicit ecall. 4. The Implicit ecall invokes the Intel® SGX PCL flow. 5. Intel® SGX PCL unseals the sealed blob to get the decryption key. 6. Intel® SGX PCL decrypts the encrypted IP sections and returns the enclave to its functional state. 7. The process continues with the enclave runtime initialization flow.

Note: In the simulation mode, link libsgx_pclsim.a and not libsgx_pcl.a

Security Considerations

Not Encrypted Sections

The ISV must make sure the ELF sections in Table below do not contain the ISV IP. The encryption tool will **NOT** encrypt these sections.

Table 15 Not Encrypted Sections

Section name	Description
.shstrtab	Sections' names string table. Pointed by e_shstrndx
.note.sgxmeta	Used by the Intel® SGXSDK
.bss and .tbss	Zero initialized data
.dynamic	Section is required to construct dyn_info by function parse_dyn at elfparser.cpp
.dynsym, .dynstr, .rela.dyn, .rela.plt	Sections hold the content pointed by entries with index DT_SYMTAB, DT_STRTAB, DT_REL and DT_PLTREL in dyn_info
.plctbl, .nipx, .nipd, .niprod	Sections which contain Intel® SGX PCL

	code and data (nip stands for Non IP)
Debug only:	
.comment, .debug_abbrev, .debug_aranges, .debug_info, .debug_line, .debug_loc, .debug_ragnes, .debug_str, .symtab, .strtab, .gnu_version_d	These sections remain plain text to enable / ease debugging.

Writable Sections and Segments

The Intel® SGX PCL encryption tool sets the Writable bit in the section flags of the encrypted ELF sections and in the segment flags of ELF segments that include the encrypted ELF sections. As a result, all pages that belong to such ELF segments or 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 OpenSSL1.1.0g (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 OpenSSL1.1.0g 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

- Add the following code to the IP Enclave link flags:
  `-Wl,--whole-archive -l<pcl_archive_name> -Wl,--no-whole-archive`
  where <pcl_archive_name> is sgx_pcl and sgx_pclsim for the HW and simulation modes, respectively.
- Add the following stage to the build flow:

```
ifneq ($(SGX_IPLDR),0)
PCL_ENCRYPTION_TOOL := sgx_encrypt
PCL_KEY := key.bin
ifeq ($(SGX_DEBUG),1)
ENCRYPTION_TOOL_FLAGS := -d
endif
$(ENCRYPTED_ENCLAVE_NAME): $(ENCLAVE_NAME) $(PCL_ENCRYPTION_TOOL)
$(PCL_ENCRYPTION_TOOL) -i $< -o $@ -k $(PCL_KEY) $(ENCRYPTION_
TOOL_FLAGS)
endif
```

  - In the debug mode, add the '-d' option. It prevents the tool from encrypting or zeroing sections used for debugging.
- Modify the build flow so that the sealed enclave is generated from the encrypted enclave.
- 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.so"
#define SEALING_ENCLAVE_FILE_NAME "SealingEnclave.signed.so"
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.

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 2.17.0.3, Intel has updated the Intel SGX SDK for Linux 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).

Addressing Stale Data Read from Legacy xAPIC

This section describes changes introduced in version 2.17.101.1 of the Intel SGX SDK for Linux.

INTEL-SA-00657 describes an issue, named Stale Data Read from Legacy xAPIC, that affects SGX. As explained here in the associated technical paper, https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/stale-data-read-from-xapic.html, 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.

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
```

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 https://nvd.nist.gov/vuln/detail/CVE-2020-0551 and https://software.intel.com/security-software-guidance/software-guidance/load-value-injection.

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]/lib64/`
Load	`[Intel SGX SDK Install Path]/lib64/cve_2020_0551_load`
CF	`[Intel SGX SDK Install Path]/lib64/cve_2020_0551_cf`

The MITIGATION-CVE-2020-0551 make variable is used to select the desired set of libraries.

MITIGATION-CVE-2020-0551 make variable value	Which set of trusted libraries?
Not set	Unmitigated
LOAD	Load
CF	CF

See the Intel® SGX SDK SampleEnclave sample (`[Intel SGX SDK Install Path]/SampleCode/SampleEnclave`) for an example.

Note that the MITIGATION-CVE-2020-0551 make variable affects how your code is built by adding the same mitigations to this code. This is generally what is desired. However, it is possible to use a mitigation level for the code that is different from the mitigation level of the Intel® SGX SDK trusted libraries. Before describing how you can do this, the necessary tools and their options must be described.

Required tools:

- GCC 7.3 or higher.
- GNU Binutils 2.36 or higher that include the mitigation support. Until a critical mass of Linux* distributions include 2.36 or higher, Intel will post the required Binutils tools here: 01.org distribution. The latest GNU Binutils can be found here: Binutils-GNU Project - Free Software Foundation.

Required options:

	Load mitigation level	CF mitigation level
GCC	`-mindirect-branch-register` `-mfunction-return=thunk-extern`	
GNU assembler, as	`-mlfence-after-load=yes` `-mlfence-before-ret=not`	`-mlfence-before-indirect-branch=register` `-mlfence-before-ret=not`

If you use a mitigation level that is different from the mitigation level of the Intel® SGX SDK trusted libraries, you must do the following:

- Do not use the MITIGATION-CVE-2020-0551 make variable.
- Specify exactly which Intel® SGX SDK trusted libraries to use.
- Use the options in the table above as appropriate.

Create CVE-2020-0551 Mitigation enabled trusted (enclave) project

Users can refer to `[Intel SGX SDK Install Path] SampleCode/SampleEnclave` to create CVE-2020-0551 mitigation enabled new projects.

There are mainly three changes in the Makefile compared with the previous version of SampleEnclave:

- Include buildenv.mk.
- Append `MITIGATION_CFLAGS` to `Enclave_C_Flags` and `Enclave_Cpp_Flags`
- Append `MITIGATION_LDFLAGS` to `Enclave_Link_Flags`
- Replace `SGX_LIBRARY_PATH` to `SGX_TRUSTED_LIBRARY_PATH`.

Users can follow the same way to create CVE-2020-0551 mitigation enabled trust projects.

Enable Mitigation for existing trusted project

Users can refer to `[Intel SGX SDK Install Path]SampleCode/SampleEnclave` to create CVE-2020-0551 mitigation enabled new projects.

There are mainly three changes in the Makefile compared with the previous version of SampleEnclave:

- Include buildenv.mk.
- Append `MITIGATION_CFLAGS` to `Enclave_C_Flags` and `Enclave_Cpp_Flags`.
- Append `MITIGATION_LDFLAGS` to `Enclave_Link_Flags`
- Replace `SGX_LIBRARY_PATH` to `SGX_TRUSTED_LIBRARY_PATH`.

Users can follow the same way to enable the CVE-2020-0551 mitigation for the existing trust projects.

Protection Keys in SGX

The Protection Keys feature provides an additional mechanism by which 4-level paging and 5-level controls access to user-mode addresses. The protection key is located in bits 62:59 of the paging-structure entry that mapped the page containing the linear address. Software can set a mask in the `PKRU` (protection key rights for user pages) register, to either disable access or disable write to the linear address mapped by the PTE. The `PKRU` register is a 32-bit register with the following format:

For each i (0 ≤ i ≤ 15), PKRU[2i] is the access-disable bit for protection key i (ADi) and PKRU[2i+1] is the write-disable bit for protection key i (WDi).

The Protection Keys feature offers an attacker an easy mechanism to trace data accesses at a page granularity inside SGX enclaves. To mitigate this potential attack inside SGX enclaves, the `sgx_sign` provides a field `PKRU` in the enclave configuration file, which could be used to enable or disable the Protection Keys inside enclave. If the enclave is signed and loaded as Protection Keys enabled, the tRTS will set the `PKRU` register to 0 on each root ecall. This would prevent an attacker from tracing the data accesses inside enclave. The tRTS also provides two APIs for users to read and write the PKRU register inside enclave. Refer to section Enclave Configuration File and section Intel® Software Guard Extensions Helper Functions for details.

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]SampleCode`.

- The *SampleEnclave* project shows how to create an enclave.
- 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.

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 source code is shipped with an installation package of the Intel® SGX SDK in `[Intel SGX SDK Install Path]SampleCode/SampleEnclave`. A Makefile is provided to build the `SampleEnclave` on Linux.

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.

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

Starting the Intel® SGX 2.4 release, you do not need to pass and store the launch token anymore.

Before the Intel® SGX 2.4 release, 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.

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 released with Intel® SGX SDK in `[Intel SGX SDK Install Path]SampleCode/PowerTransition`. A Makefile is provided to build the sample code on Linux* OS.

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.

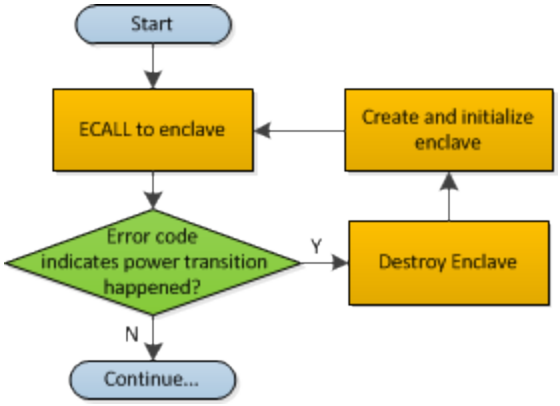

Figure 5 Power Transition Handling Flow Chart

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:

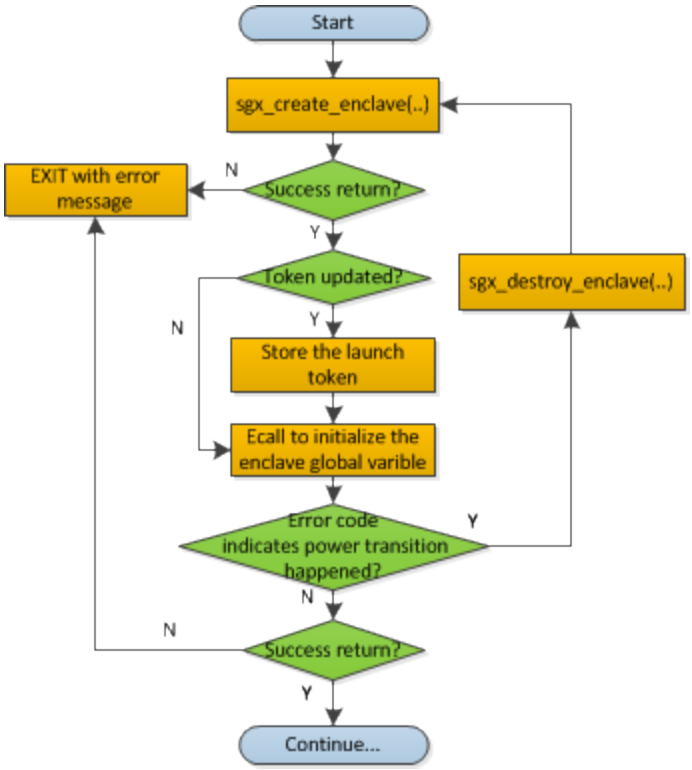

Figure 6 Enclave Initialization ECall after Enclave Creation Flow Chart

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.

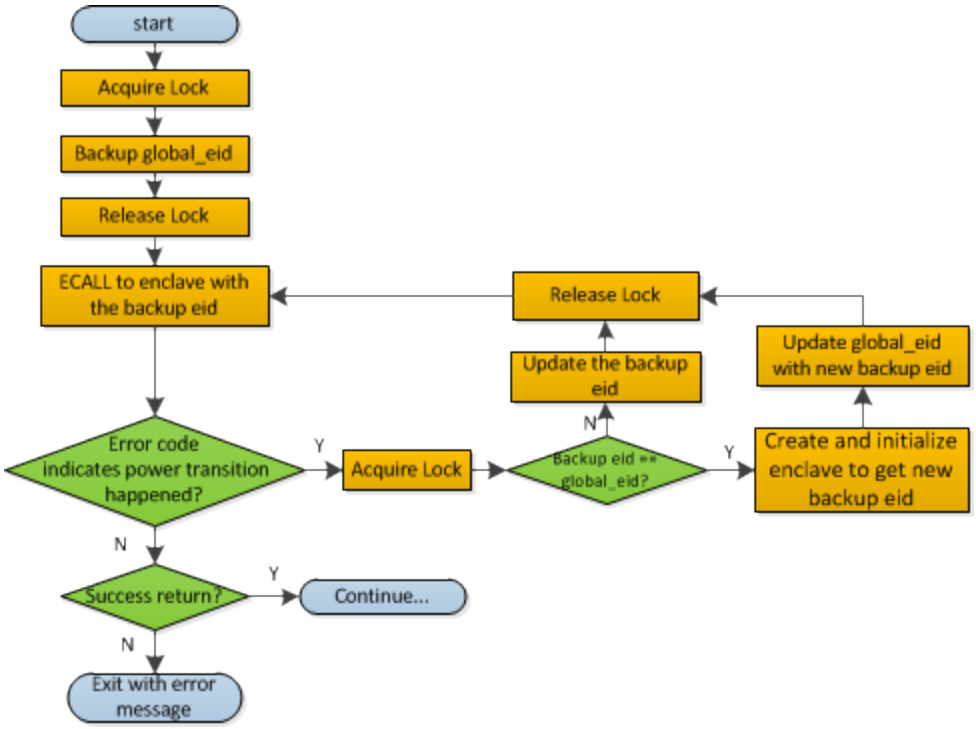

Figure 7 Regular ECALL Flow Chart

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 GCC. 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]SampleCode/Cxx11SGXDemo`. A Makefile is provided to build the Cxx11SGXDemo on Linux.

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	#include <typeinfo> #include <functional> #include <algorithm> #include <unordered_set> #include <unordered_map> #include <initializer_list> #include <tuple> #include <memory> #include <atomic> #include <mutex> #include <condition_variable> #include <map>
Classes	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 std::mutex, std::condition_variable
Compiler features	lambda expressions, auto, decltype, strongly typed enum classes, range-based for statements, static_assert, new virtual function controls, delegating constructors, variadic templates,

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

C++14 Demo

The project *Cxx14SGXDemo* is designed to illustrate some of the C++14 library features supported inside the enclave that is provided by the Intel® SGX SDK and the compiler features hat are supported by the GCC. This sample provides practical use cases for each of the C++14 features that are currently supported.

The code is shipped with the Intel SGX SDK and is located in `[Intel SGX SDK Install Path]SampleCode/Cxx14SGXDemo`. A Makefile is provided to build the *Cxx14SGXDemo* on Linux.

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++14 features inside the enclave as listed in the table below.

Table 17 Overview of C++14 features covered

Classes	std::make_unique, std::integral_constant, std::integer_sequence, std::cbegin, std::cend, std::crbegin, std::crend, std::exchange, std::is_final, std:quoted, std::equal new overload, std::mismatch new overload std::is_permutation new overload, standard user-defined literals
Compiler features	heterogeneous lookup, function return type deduction, variable template, binary literals, digit separators, generic lambdas, lambda capture expressions, attribute [[deprecated]], aggregate member initialization, alternate type deduction on declaration, relaxed constexpr restrictions

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]SampleCode/Local_Attestation` directory. A Makefile is provided to compile the project.

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 libsgx_tservice.a that is part of the SDK release.

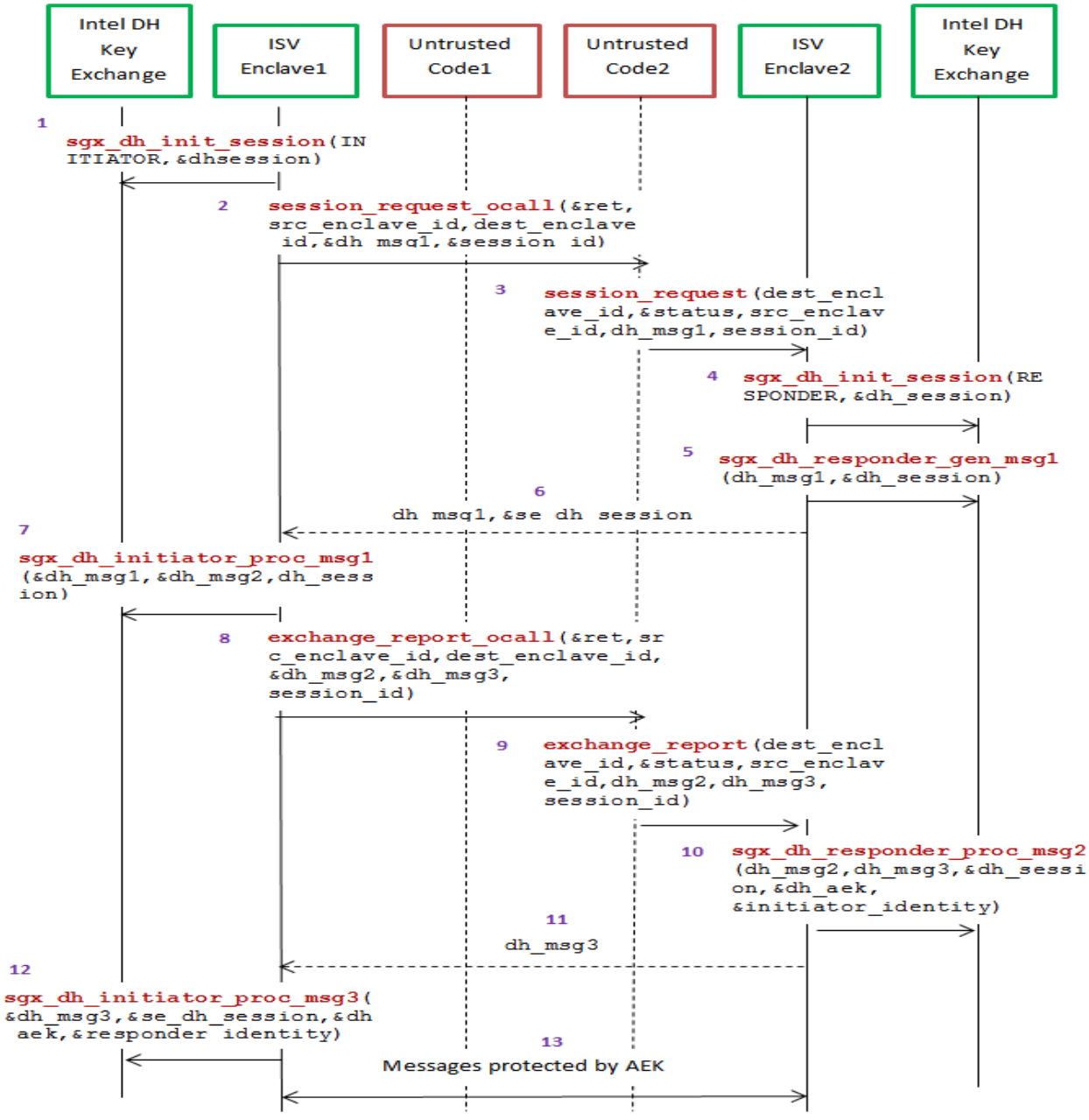

Figure 8 Local Attestation Flow with the DH Key Exchange Library

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_specin 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.

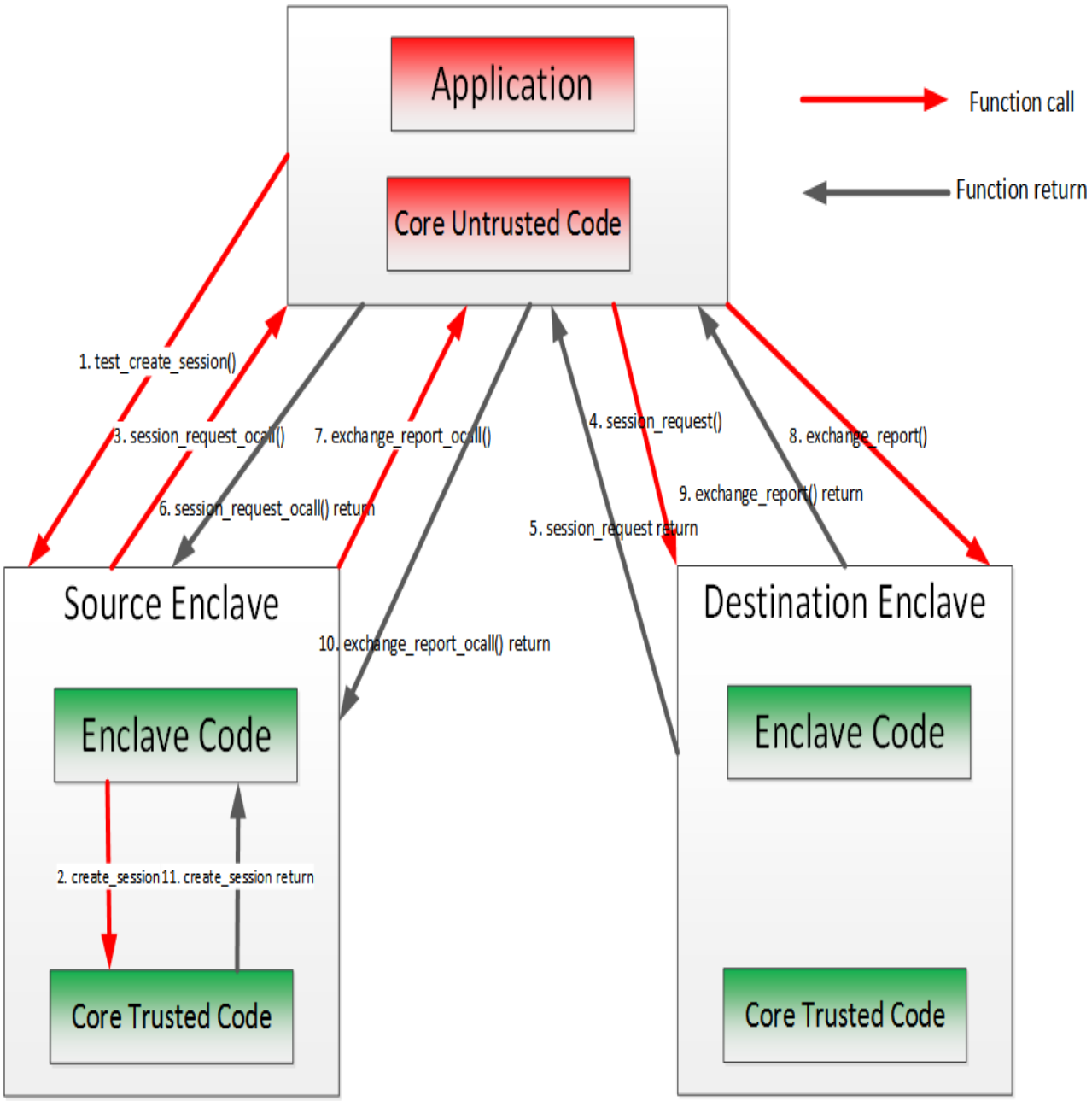

Figure 9 Secure Channel Establishment Flow with the DH Key Exchange Library

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.

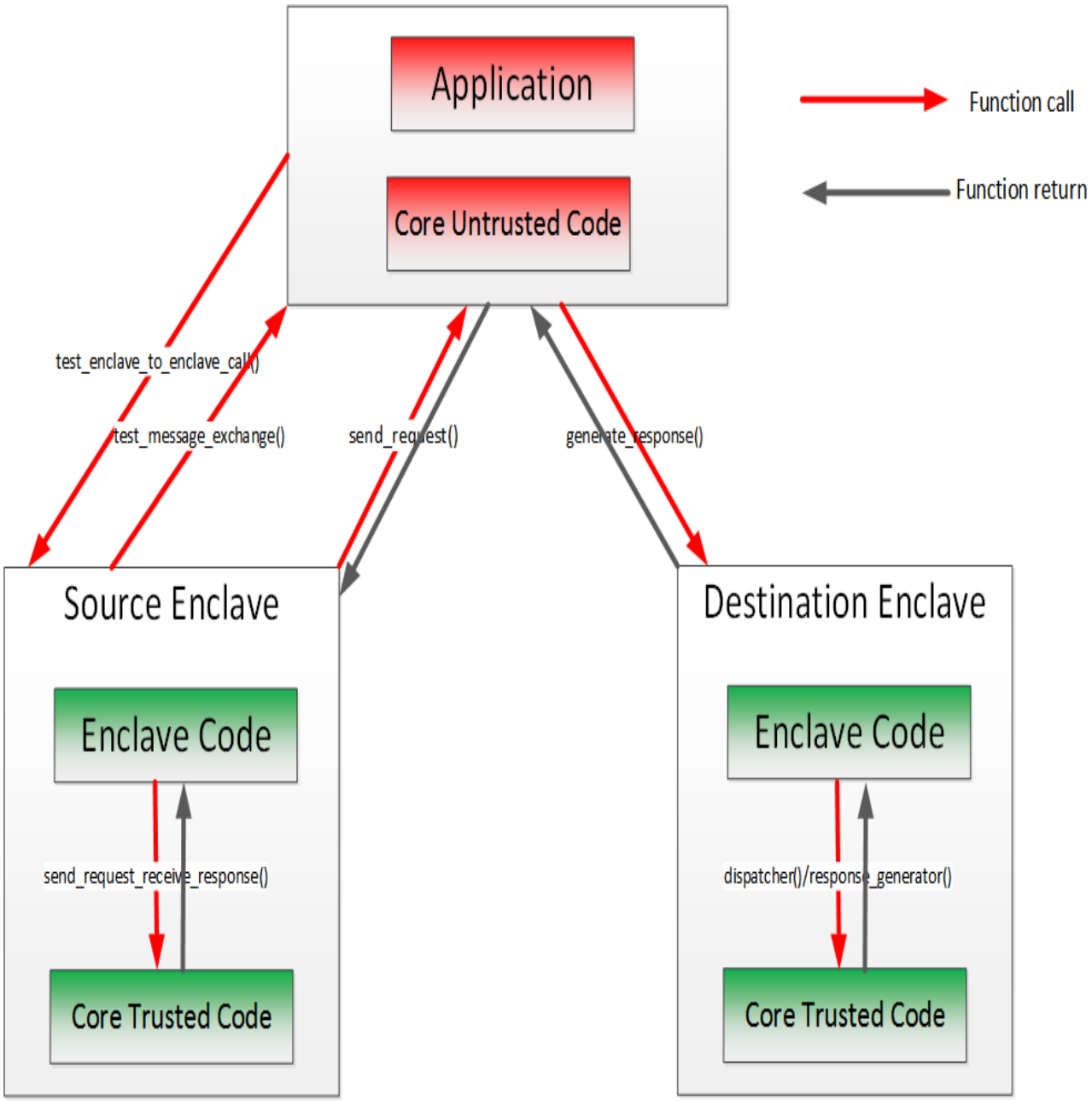

Figure 10 Secret Message Exchange Flow with the DH Key Exchange Library

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]SampleCode/RemoteAttestation`.

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

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, `libsgx_ukey_exchange.a`. 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 Shared Object, `service_provider.so`. 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.

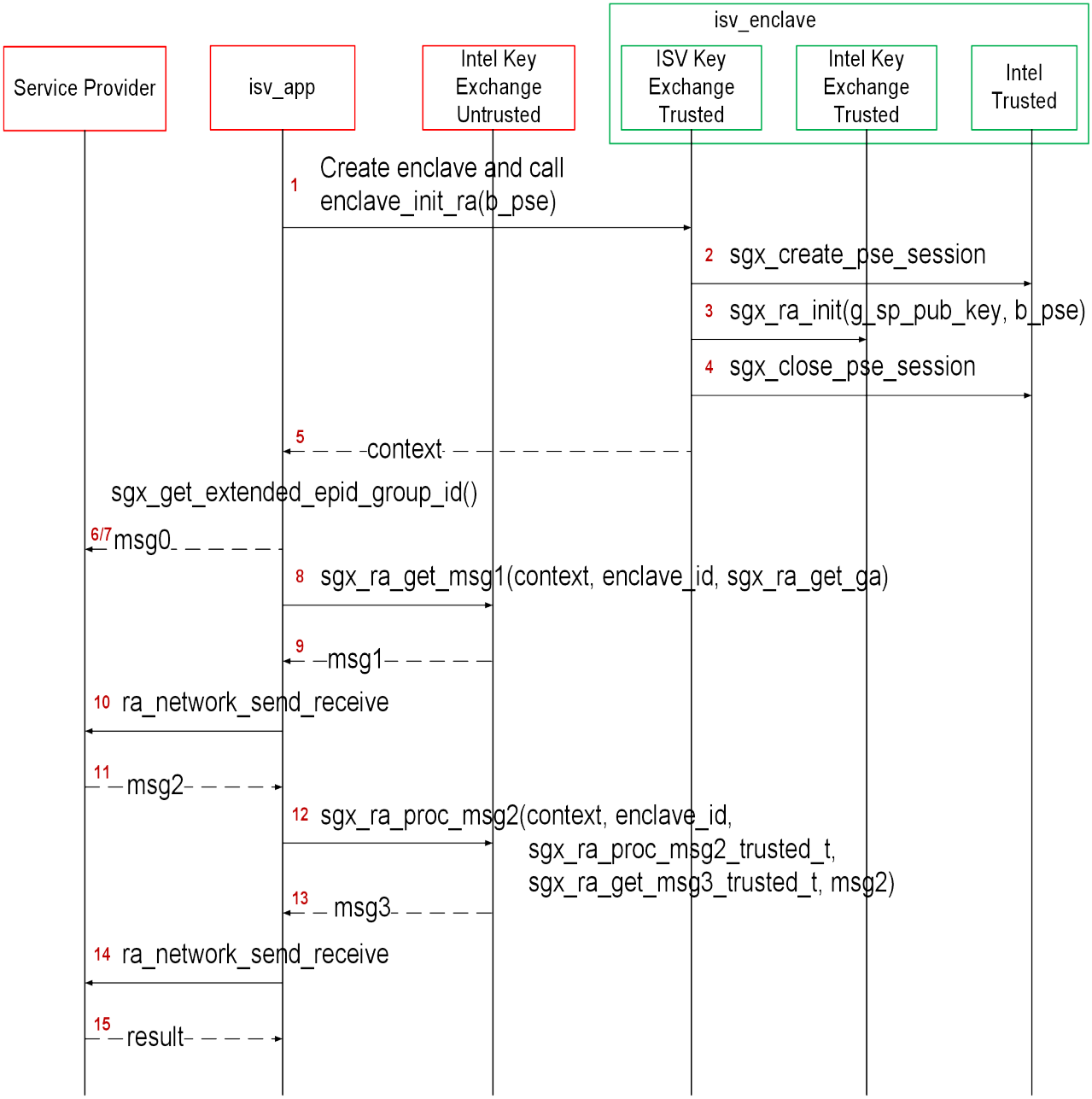

Figure 11 Remote Attestation and Trust Channel Establishment Flow

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 end-point 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 || SPID || 2-byte TYPE || 2-byte KDF-ID || SigSP(gb, ga) || 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 $S3 = CMAC_{SMK}(M)||M$ where M = ga ||PS_SECURITY_PROPERTY|| QUOTE and returns it. Platform Services Security Information is included only if the app/enclave uses Platform Services.
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 `libsample_libcrypto.so` are predictable and cryptographically weak. If we can replay `msg1` send from the `isv_app`, the sample `service_provider.` 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 `libsample_libcrypto.so` 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.

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 ECALLSs/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.

Sample Enclave for GM SMx using Intel® IPP

The project SampleEnclaveGMIPP shows you how to use the GM (Chinese National Commercial Password Algorithms) SM2, SM3, and SM4 functions inside an enclave. These functions are implemented in the Intel® IPP.

The source code is shipped with the installation package of the Intel® SGX SDK in <install-dir>SampleCode/SampleEnclaveGMIPP.

There are 3 parts of the sample project: initializing an enclave, creating an ECALL and calling GM SM2, or SM3, or SM4 functions in a specific sequence in

this ECALL, and destroying the enclave. For more details about GM SM2, SM3, and SM4 functions, refer to the Intel® IPP official page.

GM SM2

SM2 is a public key cryptographic algorithm based on elliptic curves that is used for key pair generation and digital signature verification. The procedure of using GM SM2 functions for signing and verifying is demonstrated below:

1. Create an ECC context for SM2
2. Create an SM2 key pair including a private key and a public key
3. Do user message digest for SM2 signing and verifying
4. Sign the user message using SM2 ECC context by a private key.
5. Verify the signature of the user message using SM2 ECC context by a public key
6. Remove secrets and release resources

GM SM3

SM3 is a cryptographic hash function used in digital signatures, message authentication codes, and pseudorandom number generators. The procedure of using GM SM3 functions to compute a digest of a message:

1. Initialize: initialize SM3 context
2. Update: digest the message of specified length
3. Get tag: compute the current SM3 digest value of the processed part of the message
4. Finalize: complete computation of the SM3 digest value
5. Remove secrets and release resources

GM SM4

SM4 is a block cipher algorithm used for data encryption and decryption. The procedure of using GM SM4 functions to encrypt plaintext and decrypt cipher-text by CBC (Cipher Block Chaining) mode and CTR (Counter) mode:

1. Initialize a SM4 context
2. Encrypt a plaintext to an encrypted text by CBC/CTR
3. Decrypt the encrypted text to a decrypted text by CBC/CTR
4. Compare plaintext and decrypted text
5. Remove secrets and release resources

Sample Attested TLS

The project *SampleAttestedTLS* shows you how to establish attested TLS channel based on Intel® SGX ECDSA remote attestation. Refer to `AttestedTLSREADME` and `README` in Sample project to understand the basic concept of Attested TLS channel and prerequisites.

This sample demonstrates attested TLS in two different ways:

- between two enclaves
- between an enclave application and a non enclave application

---

NOTE:

The sample based on Intel® SGX ECDSA remote attestation. Refer to Intel® SGX DCAP (Data Center Attestation Primitives) to set up environment and make sure DCAP `QuotGenerationSample` and `QuoteVerificationSample` can be ran successfully.

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

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

The untrusted libraries built for the hardware mode contain a string with the release number. The string version, which uses the library name as the prefix, is defined when the library is built. The string version consists of various parameters such as the product number, SVN revision number, build number, and so on. This mechanism ensures all untrusted libraries shipped in a given Intel® SGX PSW/SDK release have the same version number and allows quick identification of the untrusted libraries linked into an untrusted component.

For instance, `libsgx_urts.so` contains a string version `SGX_URTS_VERSION_1.0.0.0`. The last digit varies depending on the specific Intel SGX PSW/SDK release number.

Enclave Creation and Destruction

These functions are used to create or destroy enclaves:

- sgx_create_enclave
- sgx_create_enclave_ex
- sgx_create_enclave_from_buffer_ex
- sgx_create_encrypted_enclave
- sgx_destroy_enclave

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.

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.

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 Remote Attestation TLS library

This library provides two APIs that help users to verify the self-signed X.509 certificate and SGX quote in non-SGX enclave environment. It also provides support for Trusted Remote Attestation TLS library.

Note that the verification is not performed inside an SGX enclave. The APIs listed below call Intel® SGX QVL (Quote Verification Library) to verify quote, and QvE (Quote Verification Enclave) is not involved. Before using these APIs, make sure the verification environment is secure.

- tee_verify_certificate_with_evidence_host
- tee_free_supplemental_data_host

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_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

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, `libsgx_tstdc.a` 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

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

The tRTS also provides two functions to read or write the PKRU register if the enclave is configured as Protection Keys enabled:

- sgx_rdpkru
- sgx_wrpkru

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

Note:

If the exception handlers can not handle the exceptions, `abort()` is called. `abort()` makes the enclave unusable and generates another exception.

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 requiresusing 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.

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
- 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.

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 C99 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 C99 definitions within an enclave.

Locale Functions

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

Random Number Generation Functions

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

String Functions

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

Abort Function

The `abort()` function is supported within an enclave but has 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.

atexit_Function

The `atexit()` function is supported inside enclaves but only under specific circumstances. On platforms that support SGX2, the tRTS invokes the global destructors as well as function registered with the `atexit()` function when enclave is destroyed.

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 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	sgx_thread_	

	mutex_init	
	sgx_thread_ mutex_destroy	
	sgx_thread_ mutex_lock	sgx_thread_wait_untrusted_ event_ocall
	sgx_thread_ mutex_trylock	
	sgx_thread_ mutex_unlock	sgx_thread_set_untrusted_ event_ocall
Reader/Writer Lock Synchronization	sgx_thread_ rwlock_init	
	sgx_thread_ rwlock_destroy	
	sgx_thread_ rwlock_rdlock	sgx_thread_wait_untrusted_ event_ocall
	sgx_thread_ rwlock_wrlock	sgx_thread_wait_untrusted_ event_ocall
	sgx_thread_ rwlock_tryrdlock	
	sgx_thread_ rwlock_trywrlock	
	sgx_thread_ rwlock_unlock	sgx_thread_set_untrusted_ event_ocall sgx_thread_set_multiple_untrusted_events_ocall
Condition Variable Synchronization	sgx_thread_cond_ init	
	sgx_thread_cond_ destroy	
	sgx_thread_cond_ wait	sgx_thread_wait_untrusted_ event_ocall sgx_thread_setwait_untrusted_ events_ocall
	sgx_thread_cond_ signal	sgx_thread_set_untrusted_ event_ocall

	sgx_thread_cond_broadcast	sgx_thread_set_multiple_untrusted_events_ocall
Thread Management	sgx_thread_self	
	sgx_thread_equal	

NOTE:
Use POSIX aligned pthread functions when you need to synchronize threads execution.

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 <mbusafecrt.h> to use them.

See Supported C Secure Functions for a list of supported secure functions definitions within an enclave.

GCC* Built-in Functions

GCC* provides built-in functions with optimization purposes. When GCC recognizes a built-in function, it will generate the code more efficiently by leveraging its optimization algorithms. GCC always treats functions with `__builtin_` prefix as built-in functions, such as `__bultin_malloc`, `__builtin_strncpy`, and so on. In many cases, GCC tries to use the built-in variant for standard C functions, such as `memcpy`, `strncpy`, and `abort`. A call to the C library function is generated unless the `-fno-builtin` compiler option is specified.

GCC optimizes built-in functions in certain cases. If GCC does not expand the built-in function directly, it will call the corresponding library function (without the `__builtin_` prefix). The trusted C library must supply a version of the functions to ensure the enclave is always built correctly.

The trusted C library does not contain any function considered insecure (for example, `strcpy`) or that may contain illegal instructions in Intel SGX (for

example, `fprintf`). However, the ISV should be aware that GCC may introduce security risks into an enclave if the compiler inlines the code corresponding to an insecure built-in function. In this case, the ISV may use the `-fno-builtin` or `-fno-builtin-function` options to suppress any unwanted built-in code generation.

See Unsupported GCC* Built-in Functions within an enclave for a list of unsupported GCC built-ins.

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 predetermined 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 on how to use `setjmp`/`longjmp` securely.

Reserved Memory Functions

Intel(R) SGX SDK allows users to configure a reserved memory area for special usage, such as JIT support. The memory is allowed to be configured or changed to executable. See Enclave Configuration File for details. To manage the reserved memory the `sgx_tstdc` library provides the following functions to query the memory information, allocate and deallocate the memory, change the memory protection.

- sgx_get_rsrv_mem_info
- sgx_alloc_rsrv_mem
- sgx_alloc_rsrv_mem_ex
- sgx_free_rsrv_mem
- sgx_tprotect_rsrv_mem

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 (including STL) that conforms to the C++14 standard. However, a different version of this library will be linked depending on the Makefile you use to develop enclaves. The newer Makefile version uses `sgx_tcxx.a` by default, which has been ported from libc++ and supports most C++14 features. On the other hand, previous Makefile versions use `sgx_tstdcxx.a`. This library was ported from STLport but provides limited C++11 support. If you update an enclave project and want access to all the supported C++14 features, you need to manually add the new C++ trusted library. If `sgx_tstdcxx.a` is present, replace it with `sgx_tcxx.a`. Otherwise, simply add `sgx_tcxx.a` to the linker dependency list of the Makefile.

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. Functions depending on a locale library;
2. 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.

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

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.

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

Trusted Remote Attestation TLS library

These functions allow end user to create attested TLS channel that are based on Intel® SGX ECDSA remote attestation. You can call these APIs to generate a self-signed X.509 certificate with embedded SGX quote, verify the self-signed X.509 certificate and SGX quote.

Note that this library depends on Intel®SGX ECDSA remote attestation. Before using these APIs, make sure your system can run Intel® SGX DCAP.

- tee_get_certificate_with_evidence
- tee_free_certificate
- tee_verify_certificate_with_evidence
- tee_free_supplemental_data

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:

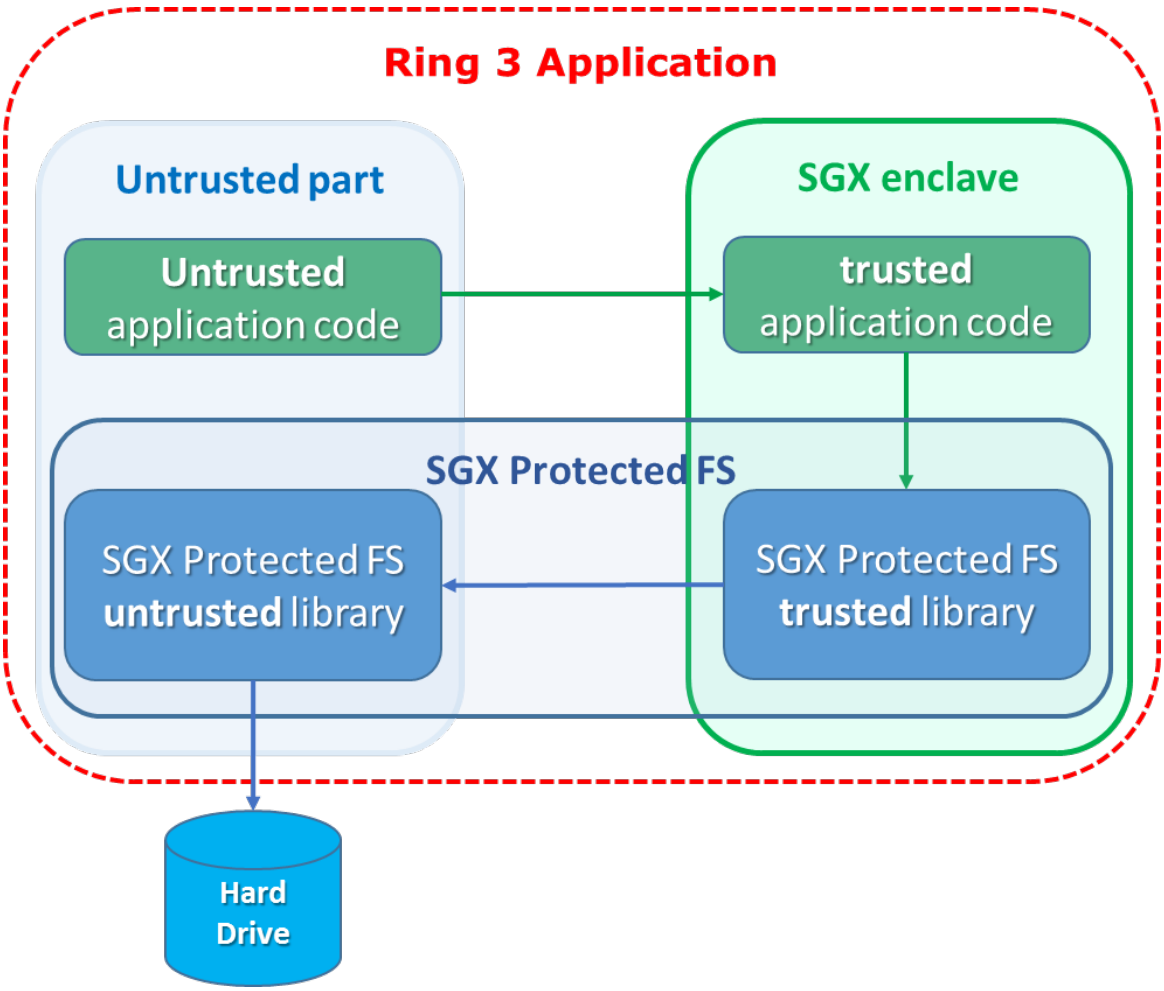

Figure 12 Protected File System Layout

To use the Intel SGX Protected File System libraries:

1. The enclave must be linked with `libsgx_tprotected_fs.a`
2. The application must be linked with `libsgx_uprotected_fs.a`
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 los). 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-2.5:

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

Do the following to enable TCMalloc in Intel® SGX:

1.Set the enclave `HeapMaxSize` equal or larger than `0x900000` in `Enclave.config.xml`.

For example:

```
<HeapMaxSize>0x900000</HeapMaxSize>
```

2.Add "`-Wl,--whole-archive -lsgx_tcmalloc -Wl,--no-whole-archive`" into enclave linking options in the Makefile.

For example:

```
Enclave_Link_Flags := $(SGX_COMMON_CFLAGS) -Wl,--no-
undefined -nostdlib -nodefaultlibs -nostartfiles -
L$(SGX_LIBRARY_PATH) \
-Wl,--whole-archive -l$(Trts_Library_Name) -Wl,--no-
whole-archive \
-Wl,--whole-archive -lsgx_tcmalloc -Wl,--no-whole-
archive \
-Wl,--start-group -lsgx_tstdc -lsgx_tstdcxx -l$(Crypto_
Library_Name) -l$(Service_Library_Name) -Wl,--end-group
\
-Wl,-Bstatic -Wl,-Bsymbolic -Wl,--no-undefined \
-Wl,-pie,-eenclave_entry -Wl,--export-dynamic \
-Wl,--defsym,__ImageBase=0 \
-Wl,--version-script=Enclave/Enclave.lds
```

NOTE:

The flags "`-Wl,--whole-archive -lsgx_tcmalloc -Wl,--no-whole-archive`" must be inserted before "`-Wl,--start-group -lsgx_tstdc -lsgx_tstdcxx -Wl,--end-group`".

Otherwise, the enclave build will fail.

Switchless Calls Library

The untrusted portion of the Switchless Calls is integrated into the uRTS library. The trusted part is provided by the `libsgx_tswitchless.a` 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 and linking it with the `libsgx_tswitchless.a` library.

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 `libsgx_pcl.a` and `libsgx_pclsim.a` libraries in HW and Simulation modes respectively. 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.

pthreads

The Intel® Software Guard Extensions (Intel® SGX) SDK includes a trusted version of the pthreads library. The library is named `sgx_pthread`, and can only be used inside an enclave.

With the trust pthreads library, a single Enclave process can contain multiple threads executing the same program.

These threads share the same global memory inside Enclave(data and heap segments) but each thread has its own stack and TCS. You need to configure enough `<TCSNum>` in Enclave Configuration File; otherwise `pthread_create()` fails.

The list below contains APIs that the Intel® SGX pthread library supports. These API interfaces are aligned with the POSIX standard.

- `pthread_create(pthread_t *threadp, const pthread_attr_t *attr, void*(*start_routine)(void*), void*arg)`
  Description: start a new thread in the calling process.

  NOTE
  The `attr` is not supported inside the Enclave, so the new thread will be created with PTHREAD_CREATE_JOINABLE.

- `pthread_exit(void *retval)`

  Description: terminate the calling thread.

- `pthread_join(pthread_t thread, void **retval)`

  Description: join with a terminated thread.

  NOTE

This function returns directly when called inside Enclave's destructor process on the Intel® SGX 2.0 platform. URTS syncs and recycles running threads when destroying the Enclave.

- `pthread_self(void)`

  Description: obtain ID of the calling thread.

- `pthread_mutex_init(pthread_mutex_t *mutexp, const pthread_mutexattr_t *attr)`
  Description: initialize the mutex referenced by `mutex`.

  NOTE
  The `attr` is not supported inside the Enclave.

- `pthread_mutex_destroy(pthread_mutex_t *mutexp)`

  Description: destory the mutex.

- `pthread_mutex_lock(pthread_mutex_t *mutexp)`

  Description: lock the mutex.

- `pthread_mutex_trylock(pthread_mutex_t *mutexp)`

  Description: lock the mutex. If the mutex is currently locked by other thread, it returns immediately.

- `pthread_mutex_unlock(pthread_mutex_t *mutexp)`

  Description: unlock the mutex.

- `pthread_rwlock_init(pthread_rwlock_t *rwlockp, const pthread_rwlockattr_t *attr)`
  Description: initialize the read-write lock referenced by `rwlockp`.

  NOTE
  The `attr` is not supported inside the Enclave.

- `pthread_rwlock_destroy(pthread_rwlock_t *rwlockp)`

  Description: destroy the read-write lock object.

- `pthread_rwlock_rdlock(pthread_rwlock_t *rwlockp)`

  Description: acquire a reader lock.

- `pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlockp)`

  Description: attempt to acquire a reader lock. If the corresponding writer lock is currently held by another thread, it returns immediately.

- `pthread_rwlock_wrlock(pthread_rwlock_t *rwlockp)`

  Description: acquire a writer lock.

- `pthread_rwlock_trywrlock(pthread_rwlock_t *rwlockp)`

  Description: attempt to acquire a writer lock. If the corresponding reader or writer lock is currently held by another thread, it returns immediately.

- `pthread_rwlock_unlock(pthread_rwlock_t *rwlockp)`

  Description: release the reader or writer lock held by the thread.

- `pthread_cond_init(pthread_cond_t *condp, const pthread_condattr_t *attr)`
  Description: initialize a condition variable.

  NOTE
  The `attr` is not supported inside the Enclave.

- `pthread_cond_destory(pthread_cond_t *condp)`

  Description: destory a condition variable.

- `pthread_cond_wait(pthread_cond_t *condp, pthread_mutex_t *mutexp)`
  Description: wait for a condition.

- `pthread_cond_signal(pthread_cond_t *condp)`

  Description: signal a condition.

- `pthread_cond_broadcast(pthread_cond_t *condp)`

  Description: broadcast a condition.

- `pthread_key_create(pthread_key_t *key, void (*destructor)(void*))`
  Description: create a thread-specific data key.

- `pthread_key_delete(pthread_key_t key)`

  Description: delete a thread-specific data key.

- `pthread_get_specific(pthread_key_t key)`
  Description: manage thread-specific data.
- `pthread_set_specific(pthread_key_t key, const void *data)`
  Description: manage thread-specific data.
- `pthread_equal(pthread_t t1, pthead_t t2)`
  Description: compare thread IDs.
- `pthread_once(pthread_once_t *once_control, void (*init_routine)(void*))`
  Description: dynamic package initialization.

To enable trust pthreads in Intel® SGX, add `"-lsgx_pthread"` into enclave linking options in the Makefile.

Intel® SGX OpenMP Library

OpenMP (Open Multi-Processing) API is a portable, scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications in C/C++ and Fortran. The Intel® Software Guard Extensions Software Development Kit (Intel® SGX SDK) includes a trusted version of OpenMP library for C/C++, named `libsgx_omp.a`. The Intel® SGX OpenMP library is a customized version based on LLVM* OpenMP* API version 5.0.

The official OpenMP contains a set of directives and a set of runtime library routines. Not all the directives and runtime routines are supported by the Intel® SGX environment. The Intel® SGX OpenMP library provides a specific subset of the directives and runtime library routines compared to the official OpenMP. See Supported OpenMP Directives and Runtime Routines for the supported list inside enclave. You can also find a list for the features that are natively unsupported by the Intel® SGX environment at Unsupported OpenMP Directives and Runtime Routines. Not listed features are not validated and are not recommended to be used.

NOTE

Intel® SGX OpenMP depends on the Intel® SGX pthreads library for thread management. As one Intel® SGX thread is designed to bind one TCS, the `TCSNum` in the enclave's configuration file should be enough. And it is also recommended to configure enough heap and stack for the OpenMP enabled

---

enclaves. Otherwise, Intel® SGX OpenMP may throw exception because of resource shortage.

To enable Intel® SGX OpenMP library in an enclave, follow the steps below to update the Makefile and the enclave EDL file:

1. Add `'-fopenmp'` option to the enclave compiling options in the Make-file.
2. Add `'-lsgx_pthread -lsgx_omp'` to the enclave linking options in the Makefile.
3. Add `'from "sgx_pthread.edl" import *;'` to the enclave EDL file.

Supported OpenMP Directives and Runtime Routines

The Intel® SGX OpenMP supports the following OpenMP directives:

Category	Syntax
Parallel construct	`#pragma omp parallel`

Work-sharing constructs	`#pragma omp for/do`
	`#pragma omp sections`
	`#pragma omp single`
	`#pragma omp master`

Data sharing attributes clauses	`shared, private, default, firstprivate, lastprivate, reduction`
Synchronization clauses	`critical, atomic, ordered, collapse, barrier, nowait`
initialization clauses	`firstprivate, lastprivate, threadprivate,`
Data copying clauses	`copyin, copyprivate`
If-control clause	`if`
Others	`flush, master`

Supported runtime routines:

- `omp_get_num_threads(void)`
- `omp_get_dynamic(void)`

- omp_get_nested (void)
- omp_get_max_threads(void)
- omp_get_thread_num (void)
- omp_get_num_procs (void)
- omp_in_parallel (void)
- omp_in_final (void)
- omp_get_active_level (void)
- omp_get_level (void)
- omp_get_ancestor_thread_num(int)
- omp_get_team_size(int);
- omp_get_thread_limit(void)
- omp_get_max_active_levels(void)
- omp_get_max_task_priority(void)
- omp_get_num_teams(void)
- omp_get_team_num(void)
- omp_init_lock(omp_lock_t *)
- omp_set_lock(omp_lock_t *)
- omp_unset_lock(omp_lock_t *)
- omp_destroy_lock(omp_lock_t *)
- omp_test_lock(omp_lock_t *)
- omp_init_nest_lock(omp_nest_lock_t *)
- omp_set_nest_lock(omp_nest_lock_t *)
- omp_unset_nest_lock(omp_nest_lock_t *)
- omp_destroy_nest_lock(omp_nest_lock_t *)
- omp_test_nest_lock(omp_nest_lock_t *)
- omp_test_nest_lock(omp_nest_lock_t *)
- omp_set_num_threads(int)
- omp_set_dynamic(int)
- omp_set_nested(int)
- omp_set_max_active_levels(int)

- `omp_set_schedule(omp_sched_t, int)`

NOTE

Certain syntax settings and runtime routines behavior may differ from the standard OpenMP behavior. For example, a thread number. You can use `#pragma omp parallel num_threads(N)` or `omp_set_num_threads (N)` to configure the working threads number. If the desired value N is too big. it may not be effective. Intel® SGX OpenMP will take a proper value instead to avoid the resource shortage exception.

Unsupported OpenMP Directives and Runtime Routines

The Intel® SGX OpenMP does not support the OpenMP directives below:

Category	Syntax
Thread Affinity	`proc_bind`
Thread cancellation	`cancel, cancellation point`
Offloading support	`target, target data, device, map, array …`

Unsupported runtime routines:

- `omp_get_wtime(void)`
- `omp_get_wtick(void)`
- `omp_set_default_device(int)`
- `omp_is_initial_device(void)`
- `omp_get_num_devices(void)`
- `omp_get_initial_device(void)`
- `omp_target_alloc(size_t, int)`
- `omp_target_free(void *, int)`
- `omp_target_is_present(void *, int)`
- `omp_target_memcpy(void *, void *, size_t, size_t, size_t, int, int)`
- `omp_target_memcpy_rect(void *, void *, size_t, int, const size_t *, const size_t *, const size_t *, const size_t *, const size_t *, int, int)`
- `omp_target_associate_ptr(void *, void *, size_t, size_t, int)`
- `omp_target_disassociate_ptr(void *, int)`
- `omp_get_device_num(void)`

- `kmp_set_warnings_on(void)`
- `kmp_set_warnings_off(void)`
- `kmp_set_affinity(kmp_affinity_mask_t *)`
- `kmp_get_affinity(kmp_affinity_mask_t *)`
- `kmp_get_affinity_max_proc(void)`
- `kmp_create_affinity_mask(kmp_affinity_mask_t *)`
- `kmp_destroy_affinity_mask(kmp_affinity_mask_t *)`
- `kmp_set_affinity_mask_proc(int, kmp_affinity_mask_t *)`
- `kmp_unset_affinity_mask_proc(int, kmp_affinity_mask_t *)`
- `kmp_get_affinity_mask_proc(int, kmp_affinity_mask_t *)`
- `omp_get_proc_bind(void)`
- `omp_get_num_places(void)`
- `omp_get_place_num_procs(int)`
- `omp_get_place_proc_ids(int, int *)`
- `omp_get_place_num(void)`
- `omp_get_partition_num_places(void)`
- `omp_get_partition_place_nums(int *)`
- `omp_control_tool(int, int, void*)`
- `omp_set_affinity_format(char const *)`
- `omp_get_affinity_format(char *, size_t)`
- `omp_display_affinity(char const *)`
- `omp_capture_affinity(char *, size_t, char const *)`

Deep Neural Network Library

The Intel® Software Guard Extensions (Intel® SGX) SDK includes a trusted version of the Intel® Deep Neural Network Library (DNNL) library. The library is named `libsgx_dnnl.a`, and can only be used inside an enclave.

All the Intel® DNNL standard APIs are supported inside the Intel® SGX DNNL.

Intel® SGX DNNL is ported based on Intel® DNNL V1.1.1. This library is built with following configurations:

- The library is built as an static library.
- The library is built with CPU engine, the CPU engine is configured to use OpenMP.

To build the Intel® SGX DNNL library, follow the steps below:

1. Download Intel® SGX source codes.
2. Enter `linux-trunk/external/dnnl/`
3. Execute `make` command. The Makefile will download Intel® DNNL V1.1.1 source codes and patch the Intel® SGX specific patch automatically.

To install Intel® SGX DNNL library, follow the steps below:

1. Copy Intel® SGX DNNL lib to the Intel® SGX SDK installation directory.
   ```
   $cp "./sgx_dnnl/lib/libsgx_dnnl.a" "$(SGX_SDK)
   /lib64"
   ```
2. Copy Intel® SGX DNNL header file to the Intel® SGX SDK installation directory.
   ```
   $cp "./sgx_dnnl/include/*" "$(SGX_SDK)/include"
   ```

To enable Intel® SGX DNNL in Enclave, do the following:

1. On Intel® SGX1 platform, set `<ReservedMemExecutable>` to `1` in the Enclave Configuration File.
2. Add `"-lsgx_pthread -lsgx_omp -lsgx_dnnl"` into enclave linking options in Makefile.
3. Add `'from "sgx_pthread.edl" import *;'` to the enclave EDL file.

Intel® SGX Protobuf Library

Protocol Buffers (a.k.a., protobuf) are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. You can find documentation on protobuf on the Google Developers site. The Intel® Software Guard Extensions Software Development Kit (Intel® SGX SDK) includes a trusted version of protobuf library for C/C++, named `libsgx_protobuf.a`. The Intel® SGX Protobuf library is a customized version based on Google's Protocol Buffers v3.14.0.

To enable Intel® SGX Protobuf library in an enclave

1. Follow the steps below to update the Makefile:
   1. Add `'-I$(SGX_SDK)/include/tprotobuf'` to the enclave header files options.
   2. Add `'-lsgx_protobuf'` to the enclave linking options.
2. Define `printf` in the enclave for the SGX Protobuf logging module.

Unsupported protobuf Field types, Classes and Functions

Unsupported Field types:

- `Duration`
- `Timestamp`
- `Service`
- `Api`
- `Method`
- `Empty`
- `FieldMask`
- `Value`
- `Mixin`

Unsupported classes:

- `GzipInputStream`
- `GzipOutputStream`
- `FileInputStream`
- `FileOutputStream`
- `IstreamInputStream`
- `OstreamOutputStream`
- `TimeUtil`

Unsupported Functions:

- `SerializeToFileDescriptor(int) const`
- `ParseFromFileDescriptor (int)`
- `bool SerializeToOstream(ostream*) const`
- `bool ParseFromIstream(istream*)`
- `operator<<(std::ostream&, const uint128&)`
- `operator<<(std::ostream&, const Status uint128&)`
- `operator<<(std::ostream&, StringPiece uint128&)`

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

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.

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 [deprecated]

Pointer to an sgx_launch_token_t object used to initialize the enclave to be created (deprecated)

launch_token_updated [deprecated]

This parameter is deprecated.

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' 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 occured. 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 undefined symbol.

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_PCL_ENCRYPTED

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

SGX_ERROR_FEATURE_NOT_SUPPORTED

Requested feature is not supported. Possible features are KSS, EDMM.

SGX_ERROR_MEMORY_MAP_FAILURE

Failed to reserve memory for the enclave.

SGX_ERROR_UNEXPECTED

Unexpected error is detected.

Description

The `sgx_create_enclave` function loads and initializes the enclave using the enclave file name.

If the enclave is valid, the function returns a value of `SGX_SUCCESS`. The enclave ID (handle) is returned via the enclave_id parameter.

The library `libsgx_urts.so` 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 `libsgx_urts_sim.so` 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. 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.

Notes on signal handling: If the Linux* kernel you use does not support the vDSO function to enter the enclave, URTS installs a signal handler for

SIGSEGV, SIGILL, SIGFPE, SIGBUS, SIGTRAP during the first call, after the enclave is successfully loaded and initialized (EINIT is done) . Its purpose is to handle signals caused by exceptions inside enclaves or at EENTER, ERESUME instructions used to enter enclaves. The handler invokes previously installed signal handlers to handle those signals raised in other situations unrelated to enclaves. If an application installs signal handler(s) for these signals after `sgx_create_enclave` is called, the newly installed handler(s) should also forward those signals to previously installed handler in order for enclave related exceptions to be handled properly by the URTS handler. If the Linux* kernel you use supports the vDSO function to enter the enclave, the vDSO function intercepts any exceptions that occur when running the enclave.

Requirements

Header	`sgx_urts.h`
Library	`libsgx_urts.so` or `libsgx_urts_sim.so` (simulation)

sgx_create_enclave_ex

Loads the enclave using its file name.

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

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.

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 codeor 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-debig 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 [deprecated]

Pointer to an sgx_launch_token_t object used to initialize the enclave to be created (deprecated).

launch_token_updated [deprecated]

This parameter is deprecated.

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' 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 priviledge.

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 undefined symbol.

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_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_MEMORY_MAP_FAILURE

Failed to reserve memory for the enclave.

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.

Notes on signal handling: If the Linux* kernel you use does not support the vDSO function to enter the enclave, URTS installs a signal handler for SIGSEGV, SIGILL, SIGFPE, SIGBUS, SIGTRAP during the first call of the `sgx_create_enclave_ex` function, after the enclave is successfully loaded and initialized (EINIT is done). Its purpose is to handle signals caused by exceptions inside enclaves or at EENTER, ERESUME instructions used to enter enclaves. The handler invokes previously installed signal handlers to handle

those signals raised in other situations unrelated to enclaves. If an application installs signal handler(s) for these signals after the call of `sgx_create_enclave_ex`, the newly installed handler(s) should also forward those signals to the previously installed handler ifor enclave related exceptions to be handled properly by the URTS handler. If the Linux* kernel you use supports the vDSO function to enter the enclave, the vDSO function intercepts any exceptions that occur when running the enclave.

Requirements

Header	`sgx_urts.h`
Library	`libsgx_urts.so` or `libsgx_urts_sim.so` (simulation)

sgx_create_encrypted_enclave

Loads the encrypted enclave using its file name. Enables Intel® SGX PCL.

Syntax

```
sgx_status_t sgx_create_encrypted_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,
    uint8_t* sealed_key
);
```

Parameters

file_name [in]

Name or full path to the enclave image.

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 [deprecated]

Pointer to an sgx_launch_token_t object used to initialize the enclave to be created (deprecated).

launch_token_updated [deprecated]

This parameter is deprecated.

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.

sealed_key [in]

Pointer to a sealed Intel® SGX PCL decryption key.

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’ 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 file is not found or you have no privilege to access the 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 needed for this operation cannot be loaded.

SGX_ERROR_NO_DEVICE

The 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

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

SGX_ERROR_MODE_INCOMPATIBLE

The 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

The request to the AE service timed out.

SGX_ERROR_SERVICE_INVALID_PRIVILEGE

The request requires some special attributes for the enclave, but does not have priviledge.

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 undefined symbol.

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_PCL_NOT_ENCRYPTED

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

SGX_ERROR_MEMORY_MAP_FAILURE

Failed to reserve memory for the enclave.

SGX_ERROR_UNEXPECTED

Unexpected error is detected.

Description

The `sgx_create_encrypted_enclave` function loads and initializes an encrypted enclave using the enclave file name, the launch token, and the sealed decryption key blob. If the launch token is incorrect, the function gets a new one and save it back to the input parameter “token”. The parameter “updated” indicates that the launch token has been updated.

If the enclave launch token and the sealed decryption key blob are valid, the function returns `SGX_SUCCESS`. The enclave ID (handle) is returned via the enclave_id parameter.

The library `libsgx_urts.so` 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`libsgx_urts_sim.a` exposes an identical interface which can only load a simulation

enclave. The simulation mode does not require the 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. A compromised loader or an 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.

Notes on signal handling: If the Linux* kernel you use does not support the vDSO function to enter the enclave, URTS installs a signal handler for SIGSEGV, SIGILL, SIGFPE, SIGBUS, SIGTRAP during the first call of the `sgx_create_encrypted_enclave` function, after the enclave is successfully loaded and initialized (EINIT is done). Its purpose is to handle signals caused by exceptions inside enclaves or at EENTER, ERESUME instructions used to enter enclaves. The handler invokes previously installed signal handlers to handle those signals raised in other situations unrelated to enclaves. If an application installs signal handler(s) for these signals after `sgx_create_encrypted_enclave` is called, the newly installed handler(s) should also forward those signals to the previously installed handler for enclave related exceptions to be handled properly by the URTS handler. If the Linux* kernel you use supports the vDSO function to enter the enclave, the vDSO function intercepts any exceptions that occur when running the enclave.

Requirements

Header	`sgx_urts.h`
Library	`libsgx_urts.so` or `libsgx_urts_sim.so` (simulation)

sgx_create_enclave_from_buffer_ex

Loads an enclave from memory.

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

Syntax

```
sgx_status_t sgx_create_enclave_from_buffer_ex(
    uint8_t *buffer,
    size_t buffer_size,
    const int debug,
```

```
    sgx_enclave_id_t *enclave_id,
    sgx_misc_attribute_t *misc_attr,
    const uint32_t ex_features,
    const void* ex_features_p[32]
);
```

Parameters

buffer [in]

Pointer to the enclave image buffer. Must not be NULL.

buffer_size [in]

Size of the enclave image buffer. Must be non-zero.

debug [in]

The valid value is 0 or 1.

0 indicates creating the enclave in a non-debug mode. An enclave created in the 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 the 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 an enclave in this mode, 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.

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 extended features to be activated during 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' 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 do not have 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 the virtual address. Create the enclave again.

SGX_ERROR_DEVICE_BUSY

Intel® SGX driver or alow 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 has timed out.

SGX_ERROR_SERVICE_INVALID_PRIVILEGE

Request requires some special attributes for the enclave, but it is not privileged.

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 undefined symbol.

This type of error typically occurs when the enclave is built.

SGX_ERROR_INVALID_MISC

The MiscSelct or MiscMask settings are not correct.

SGX_ERROR_PCL_ENCRYPTED

Enclave is encrypted but input parameters do not include the required content (for example, 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_MEMORY_MAP_FAILURE

Failed to reserve memory for the enclave.

SGX_ERROR_UNEXPECTED

Unexpected error is detected.

Description

The `sgx_create_enclave_from_buffer_ex` function loads an enclave. This function supports extended features if the input is provided in `ex_features` and `ex_features_p` parameters. Unlike the `sgx_create_enclave_ex` function, the `sgx_create_enclave_from_buffer_ex` function allows you to load an enclave from memory without using its file name.

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. You should sign the KSS enabled enclave with `EnableKSS` set to 1 in the configuration file.

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

Notes on signal handling: If the Linux* kernel you use does not support the vDSO function to enter the enclave, URTS installs a signal handler for SIGSEGV, SIGILL, SIGFPE, SIGBUS, SIGTRAP during the first call of the `sgx_create_enclave_ex` function, after the enclave is successfully loaded and initialized (EINIT is done). Its purpose is to handle signals caused by exceptions inside enclaves or at EENTER, ERESUME instructions used to enter enclaves. The handler invokes previously installed signal handlers to handle those signals raised in other situations unrelated to enclaves. If an application installs signal handler(s) for these signals after calling `sgx_create_enclave_ex`, the newly installed handler(s) should also forward those signals to the previously installed handler for enclave related exceptions to be handled properly by the URTS handler. If the Linux* kernel you use supports the vDSO function to enter the enclave, the vDSO function intercepts any exceptions that occur when running the enclave.

Requirements

Header	`sgx_urts.h`
Library	`libsgx_urts.so` or `libsgx_urts_sim.so` (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 `libsgx_urts.so` exposes this function to destroy a previously created enclave in hardware mode, while `libsgx_urts_sim.so` provides a simulative counterpart.

See more details in Loading and Unloading an Enclave.

Requirements

Header	`sgx_urts.h`
Library	`libsgx_urts.so` or `libsgx_urts_sim.so` (simulation)

sgx_get_target_info

Gets the target info of an enclave.

Syntax

```
sgx_status_t sgx_get_target_info(
    const sgx_enclave_id_t enclave_id;
    sgx_target_info_t* target_info)
);
```

Parameters

enclave_id [in]

Enclave ID or handle generated by sgx_create_enclave.

target_info [OUT]

Pointer to the sgx_target_info_t object that receives the target info of the enclave. Must be a non-NULL pointer.

See sgx_target_info_t for structure details.

Return value

SGX_SUCCESS

All outputs are generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the pointers is invalid.

SGX_ERROR_INVALID_ENCLAVE_ID

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

SGX_ERROR_UNEXPECTED

Unexpected error is detected.

Description

The `sgx_get_target_info` function getss the target info of an enclave. The enclave should have been loaded.

The library `libsgx_urts.so` exposes this function to get a previously created enclave in a hardware mode, while `libsgx_urts_sim.so` provides a simulative counterpart.

See more details in Loading and Unloading an Enclave.

Requirements

Header	`sgx_urts.h`
Library	`libsgx_urts.so` or `libsgx_urts_sim.so` (simulation)

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_ex`to generate or obtain the attestation key. Calculate quote size by Calling `sgx_get_quote_size_ex`. At last, call `sgx_get_quote_ex`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 config file(/etc/aesmd.conf) 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	`libsgx_quote_ex.so` or `libsgx_quote_ex_sim.so` (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	`sgx_uae_epid.h`
Library	`libsgx_epid.so` or `libsgx_epid_sim.so` (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_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

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	`sgx_uae_quote_ex.h`
Library	`libsgx_quote_ex.so` or `libsgx_quote_ex_sim.so` (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	`sgx_uae_epid.h`
Library	`libsgx_epid.so` or `libsgx_epid_sim.so` (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	`sgx_uae_epid.h`
Library	`libsgx_epid.so` or `libsgx_epid_sim.so` (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	`libsgx_quote_ex.so` or `libsgx_quote_ex_sim.so` (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 GID 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	`sgx_uae_epid.h`
Library	`libsgx_epid.so` or `libsgx_epid_sim.so` (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_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_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	`sgx_uae_quote_ex.h`
Library	`libsgx_quote_ex.so` or `libsgx_quote_ex_sim.so` (simulation)

sgx_get_supported_att_key_id_num

`sgx_get_suooroted_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	`sgx_uae_quote_ex.h`
Library	`libsgx_quote_ex.so` or `libsgx_quote_ex_sim.so` (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	`sgx_uae_quote_ex.h`
Library	`libsgx_quote_ex.so` or `libsgx_quote_ex_sim.so` (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	`sgx_ukey_exchange.h`
Library	`libsgx_ukey_exchange.a`

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_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

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	`sgx_ukey_exchange.h`
Library	`libsgx_ukey_exchange.a`

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	`sgx_ukey_exchange.h`
Library	`libsgx_ukey_exchange.a`

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_NETWORK_FAILURE

Network connecting or proxy setting issue was encountered.

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	`sgx_ukey_exchange.h`
Library	`libsgx_ukey_exchange.a`

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	`sgx_uae_epid.h`
Library	`libsgx_epid.so` or `libsgx_uae_service_sim.so` (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 `p_update_info` 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	`sgx_uae_epid.h`
Library	`libsgx_epid.so` or `libsgx_uae_service_sim.so` (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	`sgx_uae_epid.h`
Library	`libsgx_epid.so` or `libsgx_uae_service_sim.so` (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(
```

```
    uint8_t *p_whitelist,
    uint32_t whitelist_size
);
```

Parameters

p_whitelist [out]

A pointer to the allowlist.

whitelist_size [in]

Size of `p_whitelist`, in bytes.

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	`libsgx_launch.so` or `libsgx_launch_sim.so` (simulation)

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	`sgx_capable.h`
Library	`libsgx_capable.so` and `libsgx_capable.a`

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_UNEXPECTED

Unexpected error is detected.

Description

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

Requirements

Header	`sgx_capable.h`
Library	`libsgx_capable.so` and `libsgx_capable.a`

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.

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_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	`sgx_capable.h`
Library	`libsgx_capable.so` and `libsgx_capable.a`

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.

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_whitelist`, call `sgx_get_whitelist_size` first to get the buffer size and then allocate enough memory for the quote.

Requirements

Header	`sgx_uae_launch.h`
Library	`libsgx_launch.so` or `libsgx_launch_sim.so` (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	`sgx_uae_launch.h`
Library	`libsgx_launch.so` or `libsgx_launch_sim.so` (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	`sgx_trts.h`
Library	`libsgx_trts.a` or `libsgx_trts_sim.a` (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	`sgx_trts.h`
Library	`libsgx_trts.a` or `libsgx_trts_sim.a` (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	`sgx_trts.h`
Library	`libsgx_trts.a` or `libsgx_trts_sim.a` (simulation)

sgx_wrpkru

The `sgx_wrpkru` function is used to modify the PKRU register inside an enclave.

Syntax

```
int sgx_wrpkru (
    uint32_t val
);
```

Parameters

val [in]

The desired PKRU value.

Return value

1

Write the PKRU register successfully.

0

Fail to write the PKRU register.

Description

`sgx_wrpkru` is used to set the PKRU register with the specific input `val` using the `wrpkru` instruction inside an enclave. It works only if the enclave is loaded as Protection Keys enabled.

Requirements

Header	`sgx_trts.h`
Library	`libsgx_trts.a` or `libsgx_trts_sim.a` (simulation)

sgx_rdpkru

The `sgx_rdpkru` function can be used to read the PKRU register value inside an enclave.

Syntax

```
int sgx_rdpkru (
    uint32_t *val
);
```

Parameters

val [out]

The output PKRU value.

Return value

1

Read PKRU register successfully.

0

Fail to read the PKRU register.

Description

`sgx_rdpkru` is used to read the PKRU register value inside an enclave using the `rdpkru` instruction. It works only if the enclave is loaded as Protection Keys enabled.

Requirements

Header	`sgx_trts.h`
Library	`libsgx_trts.a` or `libsgx_trts_sim.a` (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	`sgx_trts_exception.h`
Library	`libsgx_trts.a` or `libsgx_trts_sim.a` (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	`sgx_trts_exception.h`
Library	`libsgx_trts.a` or `libsgx_trts_sim.a` (simulation)

sgx_spin_lock

The `sgx_spin_lock` function acquires a spin lock within the enclave.

Syntax

```
uint32_t sgx_spin_lock(
    sgx_spinlock_t * lock
);
```

Parameters

lock [in]

The trusted spin lock object to be acquired.

Return value

0

This function always returns zero after the lock is acquired.

Description

`sgx_spin_lock` modifies the value of the spin lock by using compiler atomic operations. If the lock is not available to be acquired, the thread will always wait on the lock until it can be acquired successfully.

Requirements

Header	`sgx_spinlock.h`
Library	`libsgx_tstdc.a`

sgx_spin_unlock

The `sgx_spin_unlock` function releases a spin lock within the enclave.

Syntax

```
uint32_t sgx_spin_unlock(
    sgx_spinlock_t * lock
);
```

Parameters

lock [in]

The trusted spin lock object to be released.

Return value

0

This function always returns zero after the lock is released.

Description

`sgx_spin_unlock` resets the value of the spin lock, regardless of its current state. This function simply assigns a value of zero to the lock, which indicates the lock is released.

Requirements

Header	`sgx_spinlock.h`
Library	`libsgx_tstdc.a`

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` ,of, or `SGX_THREAD_RECURSIVE_MUTEX_INITIALIZER` instead.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`libsgx_tstdc.a`

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	`libsgx_tstdc.a`

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	`libsgx_tstdc.a`

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	`libsgx_tstdc.a`

sgx_thread_mutex_unlock

The `sgx_thread_mutex_unlock` function unlocks a trusted mutex object within an enclave.

Syntax

```
int sgx_thread_mutex_unlock(
    sgx_thread_mutex_t * mutex
);
```

Parameters

mutex [in]

The trusted mutex object to be unlocked.

Return value

0

The mutex is unlocked successfully.

EINVAL

The trusted mutex object is invalid or it is not locked by any thread.

EPERM

The mutex is locked by another thread.

Description

Before a thread releases a mutex, it has to verify it is the owner of the mutex. If that is the case, the thread decreases the refcount by 1 and then may either continue normal execution or wakeup the first thread in the queue. Note that to ensure the state of the mutex remains consistent, the thread that is awakened by the thread releasing the mutex will then try to acquire the mutex almost as in the initial call to the `sgx_thread_mutex_lock` routine.

Requirements

Header	`sgx_thread.h sgxtstdc.edl`
Library	`libsgx_tstdc.a`

sgx_thread_rwlock_init

The `sgx_thread_rwlock_init` function initializes a trusted rwlock object within the enclave.

Syntax

```
int sgx_thread_rwlock_init(
    sgx_thread_rwlock_t *rwlock,
    const sgx_thread_rwlockattr_t *unused
);
```

Parameters

rwlock [in]

The trusted rwlock object to be initialized.

unused [in]

Unused parameter reserved for future user defined rwlock attributes. [NOT USED]

Return value

0

The rwlock is initialized successfully.

EINVAL

The trusted rwlock object is invalid. It is either NULL or located outside of enclave memory.

Description

When a thread creates a rwlock within an enclave, `sgx_thread_rwlock_init` simply initializes the various fields of the rwlock object to indicate that the rwlock is available. The results of using a rwlock in a lock or unlock operation before it has been fully initialized (for example, the function call to `sgx_thread_rwlock_init` returns) are undefined. To avoid race conditions in the initialization of a trusted rwlock, it is recommended statically initializing the rwlock with the macro `SGX_THREAD_RWLOCK_INITIALIZER`.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`libsgx_tstdc.a`

sgx_thread_rwlock_destroy

The `sgx_thread_rwlock_destroy` function destroys a trusted rwlock object within an enclave.

Syntax

```
int sgx_thread_rwlock_destroy(
    sgx_thread_rwlock_t * rwlock
);
```

Parameters

rwlock [in]

The trusted rwlock object to be destroyed.

Return value

0

The rwlock is destroyed successfully.

EINVAL

The trusted rwlock object is invalid. It is either NULL or located outside of enclave memory.

EBUSY

The rwlock is locked (either a reader or a writer lock) by another thread or it is pending threads to acquire the rwlock.

Description

`sgx_thread_rwlock_destroy` resets the rwlock, which brings it to the initial status. The function will fail if any thread holds either a reader or a writer lock or awaits the lock.

NOTE:
Attempting to acquire either a reader or a writer lock after it has destroyed results in undefined behavior. After a rwlock is destroyed, it must be re-initialized before it can be used again.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`libsgx_tstdc.a`

sgx_thread_rwlock_rdlock

The `sgx_thread_rwlock_rdlock` function acquires a reader lock on the provided rwlock object.

Syntax

```
int sgx_thread_rwlock_rdlock(
    sgx_thread_rwlock_t * rwlock
);
```

Parameters

rwlock [in]

The trusted rwlock object to be locked.

Return value

0

The reader lock is acquired.

EINVAL

The trusted rwlock object is invalid.

EDEADLK

This thread currently holds a writer lock on the rwlock object.

Description

Used to acquire a reader lock on rwlock object. The function ensures that no other threads hold a writer lock on the rwlock object. If no other threads hold a writer lock, the reader lock is acquired and the function returns successfully. If another thread is currently holding a writer lock, the function will make an OCALL to put the thread to sleep until the writer lock is released. When the writer lock is released, the thread that releases the writer lock will make an OCALL to wake pending threads, the function will return from the OCALL and try to obtain the reader lock again.

NOTE

A thread should not exit an enclave returning from a root ECALL after acquiring the ownership of a reader lock. Do not split a critical section protected by a lock across root ECALLs.

Requirements

Header	`sgx_thread.h sgx_tsrdc.edl`
Library	`libsgx_tstdc.a`

sgx_thread_rwlock_tryrdlock

The `sgx_thread_rwlock_tryrdlock` function tries to acquire a reader lock on the provided rwlock object.

Syntax

```
int sgx_thread_rwlock_tryrdlock(
    sgx_thread_rwlock_t * rwlock
);
```

Parameters

rwlock [in]

The trusted rwlock object to be locked.

Return value

0

The reader lock is acquired.

EINVAL

The trusted rwlock object is invalid.

EDEADLK

This thread currently holds a writer lock on the rwlock object.

EBUSY

Another thread currently holds a writer lock on the rwlock object.

Description

A thread may try to acquire a reader lock on rwlock object. If no other threads hold a writer lock on the rwlock object, the reader lock is acquired. If another thread is currently holding a writer lock, the function returns EBUSY.

NOTE

A thread should not exit an enclave returning from a root ECALL after acquiring the ownership of a reader lock. Do not split a critical section protected by a lock across root ECALLs.

Requirements

Header	`sgx_thread.h sgx_tsrdc.edl`
Library	`libsgx_tstdc.a`

sgx_thread_rwlock_wrlock

The `sgx_thread_rwlock_wrlock` function acquires a writer lock on the provided rwlock object.

Syntax

```
int sgx_thread_rwlock_wrlock(
    sgx_thread_rwlock_t * rwlock
);
```

Parameters

rwlock [in]

The trusted rwlock object to be locked.

Return value

0

The writer lock is acquired.

EINVAL

The trusted rwlock object is invalid.

EDEADLK

This thread currently holds a writer lock on the rwlock object.

Description

To acquire a writer lock on rwlock object, the function ensures that no other threads hold either a reader or a writer lock on the rwlock object. If no other threads hold a either a reader or a writer lock, the writer lock is acquired and the function returned successfully. If another thread currently holds either lock, the function will make an OCALL to put the thread to sleep until the lock is released. When either the writer lock or all the reader locks are released, the thread, that releases the lock, will make an OCALL to wake pending threads. The function will return from the OCALL and try to obtain the writer lock again.

NOTE

A thread should not exit an enclave returning from a root ECALL after acquiring the ownership of a writer lock. Do not split a critical section protected by a lock across root ECALLs.

If a thread currently holds a reader lock attempts to acquire a writer lock, the thread will deadlock.

Requirements

Header	`sgx_thread.h sgx_tsrdc.edl`
Library	`libsgx_tstdc.a`

sgx_thread_rwlock_trywrlock

The `sgx_thread_rwlock_trywrlock` function tries to acquire a writer lock on the provided rwlock object.

Syntax

```
int sgx_thread_rwlock_trywrlock(
    sgx_thread_rwlock_t * rwlock
);
```

Parameters

rwlock [in]

The trusted rwlock object to be locked.

Return value

0

The writer lock is acquired.

EINVAL

The trusted rwlock object is invalid.

EDEADLK

This thread currently holds a writer lock on the rwlock object.

EBUSY

Another thread currently holds a writer or a reader lock on the rwlock object.

Description

A thread may try to acquire a writer lock on rwlock object. If no other threads hold a writer lock or a writer lock on the rwlock object, the writer lock is acquired. If another thread holds a lock, the function returns EBUSY.

NOTE

A thread should not exit an enclave returning from a root ECALL after acquiring the ownership of a reader lock. Do not split a critical section protected by a lock across root ECALLs.

Requirements

Header	`sgx_thread.h sgx_tsrdc.edl`
Library	`libsgx_tstdc.a`

sgx_thread_rwlock_unlock

The `sgx_thread_rwlock_unlock` function unlocks a trusted mutex object within an enclave.

Syntax

```
int sgx_thread_rwlock_unlock(
    sgx_thread_rwlock_t * rwlock
);
```

Parameters

rwlock [in]

The trusted rwlock object to be unlocked.

Return value

0

The rwlock is unlocked successfully.

EINVAL

The trusted rwlock object is invalid.

EPERM

The rwlock is not locked by any thread.

Description

If the thread owns the writer lock, it will release the lock. Otherwise, it will release a reader lock by decreasing the reader lock refcount by 1. If releasing a writer lock or the reference count for reader locks becomes zero, indicating all reader locks, which have been released, the function will attempt to wake threads, waiting first on reader locks, then on writer locks.

NOTE

Releasing a rwlock when the thread does not hold a lock can return undefined results.

Requirements

Header	`sgx_thread.h sgxtstdc.edl`
Library	`libsgx_tstdc.a`

sgx_thread_cond_init

The `sgx_thread_cond_init` function initializes a trusted condition variable within the enclave.

Syntax

```
int sgx_thread_cond_init(
```

```
    sgx_thread_cond_t * cond,
    const sgx_thread_condattr_t * unused
);
```

Parameters

cond [in]

The trusted condition variable.

attr [in]

Unused parameter reserved for future user defined condition variable attributes. [NOT USED]

Return value

0

The condition variable is initialized successfully.

EINVAL

The trusted condition variable is invalid. It is either NULL or located outside enclave memory.

Description:

When a thread creates a condition variable within an enclave, it simply initializes the various fields of the object to indicate that the condition variable is available. The results of using a condition variable in a wait, signal or broadcast operation before it has been fully initialized (for example, the function call to `sgx_thread_cond_init` returns) are undefined. To avoid race conditions in the initialization of a condition variable, it is recommended statically initializing the condition variable with the macro `SGX_THREAD_COND_INITIALIZER`.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`libsgx_tstdc.a`

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	`libsgx_tstdc.a`

sgx_thread_cond_wait

The `sgx_thread_cond_wait` function waits on a condition variable within an enclave.

Syntax

```
int sgx_thread_cond_wait(
```

```
    sgx_thread_cond_t * cond,
    sgx_thread_mutex_t * mutex
);
```

Parameters

cond [in]

The trusted condition variable to be waited on.

mutex [in]

The trusted mutex object that will be unlocked when the thread is blocked in the condition variable.

Return value

0

The thread waiting on the condition variable is signaled by other thread (without errors).

EINVAL

The trusted condition variable or mutex object is invalid or the mutex is not locked.

EPERM

The trusted mutex is locked by another thread.

Description:

A condition variable is always used in conjunction with a mutex. To wait on a condition variable, a thread first needs to acquire the condition variable spin lock. After the spin lock is acquired, the thread updates the condition variable waiting queue. To avoid the lost wake-up signal problem, the condition variable spin lock is released after the mutex. This order ensures the function atomically releases the mutex and causes the calling thread to block on the condition variable, with respect to other threads accessing the mutex and the condition variable. After releasing the condition variable spin lock, the thread makes an OCALL to get suspended. When the thread is awakened, it acquires the condition variable spin lock. The thread then searches the condition variable queue. If the thread is in the queue, it means that the thread was already waiting on the condition variable outside the enclave, and it has been awakened unexpectedly. When this happens, the thread releases the condition variable spin lock, makes an OCALL and simply goes back to sleep.

Otherwise, another thread has signaled or broadcasted the condition variable and this thread may proceed. Before returning, the thread releases the condition variable spin lock and acquires the mutex, ensuring that upon returning from the function call the thread still owns the mutex.

NOTE

Threads check whether they are in the queue to make the Intel SGX condition variable robust against attacks to the untrusted event.

A thread may have to do up to two OCALLs throughout the `sgx_thread_cond_wait` function call.

Requirements

Header	`sgx_thread.h sgx_tstdc.edl`
Library	`libsgx_tstdc.a`

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	`libsgx_tstdc.a`

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	libsgx_tstdc.a

sgx_thread_self

The `sgx_thread_self` function returns the unique thread identification.

Syntax

```
sgx_thread_t sgx_thread_self(
    void
);
```

Return value

The return value cannot be NULL and is always valid as long as it is invoked by a thread inside the enclave.

Description

The function is a simple wrap of `get_thread_data()` provided in the tRTS, which provides a trusted thread unique identifier.

NOTE:
This identifier does not change throughout the life of an enclave.

Requirements

Header	sgx_thread.h sgx_tstdc.edl
Library	libsgx_tstdc.a

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	`sgx_thread.h sgx_tstdc.edl`
Library	`libsgx_tstdc.a`

sgx_cpuid

The `sgx_cpuid` function performs the equivalent of a cpuid() function call or intrinisic which executes the CPUID instruction to query the host processor for the information about supported features.

NOTE:
This function performs an OCALL to execute the CPUID instruction.

Syntax

```
sgx_status_t sgx_cpuid(
    int cpuinfo[4],
    int leaf
);
```

Parameters

cpuinfo [in, out]

The information returned in an array of four integers. This array must be located within the enclave.

leaf [in]

The leaf specified for retrieved CPU info.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates the parameter cpuinfo is invalid, which would be NULL or outside the enclave.

Description

This function provides the equivalent of the cpuid() function or intrinsic. The function executes the CPUID instruction for the given leaf (input). The CPUID instruction provides processor feature and type information that is returned in cpuinfo, an array of 4 integers to specify the values of EAX, EBX, ECX and EDX registers. `sgx_cpuid` performs an OCALL by invoking oc_cpuidex to get the info from untrusted side because the CPUID instruction is an illegal instruction in the enclave domain.

For additional details, see Intel® 64 and IA-32 Architectures Software Developer's Manual for the description on the CPUID instruction and its individual leafs. (Leaf corresponds to EAX in the PRM description).

NOTE

1. As the CPUID instruction is executed by an OCALL, the results should not be trusted. Code should verify the results and perform a threat evaluation to determine the impact on trusted code if the results were spoofed.
2. The implementation of this function performs an OCALL and therefore, this function will not have the same serializing or fencing behavior of executing a CPUID instruction in an untrusted domain code flow.

Requirements

Header	`sgx_cpuid.h sgx_tstdc.edl`
Library	`libsgx_tstdc.a`

sgx_cpuidex

The `sgx_cpuidex` function performs the equivalent of a `cpuid_ex()` function call or intrinisic which executes the CPUID instruction to query the host processor for the information about supported features.

NOTE:
This function performs an OCALL to execute the CPUID instruction.

Syntax

```
sgx_status_t sgx_cpuidex(
    int cpuinfo[4],
```

```
    int leaf,
    int subleaf
);
```

Parameters

cpuinfo [in, out]

The information returned in an array of four integers. The array must be located within the enclave.

leaf[in]

The leaf specified for retrieved CPU info.

subleaf[in]

The sub-leaf specified for retrieved CPU info.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates the parameter cpuinfo is invalid, which would be NULL or outside the enclave.

Description

This function provides the equivalent of the `cpuid()` function or intrinsic. The function executes the CPUID instruction for the given leaf (input). The CPUID instruction provides processor feature and type information returned in cpuinfo, an array of 4 integers to specify the values of EAX, EBX, ECX and EDX registers. sgx_cpuid performs an OCALL by invoking oc_cpuidex to get the info from untrusted side because the CPUID instruction is an illegal instruction in the enclave domain.

For additional details, see Intel® 64 and IA-32 Architectures Software Developer's Manual for the description on the CPUID instruction and its individual leafs. (Leaf corresponds to EAX in the PRM description).

NOTE

1. As the CPUID instruction is executed by an OCALL, the results should not be trusted. Code should verify the results and perform a threat evaluation to determine the impact on trusted code if the results were spoofed.
2. The implementation of this function performs an OCALL and therefore, this function will not have the same serializing or fencing behavior of executing a CPUID instruction in an untrusted domain code flow.

Requirements

Header	`sgx_cpuid.h sgx_tstdc.edl`
Library	`libsgx_tstdc.a`

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	`sgx_utils.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_utils.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_utils.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_utils.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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 occured.

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	`sgx_utils.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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 strucutre cannot be aligned.

Description

The `sgx_get_aligned_ptr` function faciliates 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. Raw_size >= sizeof(structure) + 64 + A, where A = max((desired, traditional alignment), 8)

.

Requirements

Header	`sgx_secure_align_api.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_secure_align_api.h`

Library	libsgx_tservice.a or libsgx_tservice_sim.a (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	sgx_secure_align_api.h
Library	libsgx_tservice.a or libsgx_tservice_sim.a (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	`sgx_tseal.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_tseal.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (simulation)

sgx_get_encrypt_txt_len

The `sgx_get_encrypt_txt_len` function is a helper function for the seal library which should be used to calculate the minimum number of bytes to allocate for decrypted data returned by the sgx_unseal_data function.

Syntax

```
uint32_t sgx_get_encrypt_txt_len(
```

```
    const sgx_sealed_data_t *p_sealed_data
);
```

Parameters

p_sealed_data [in]

Pointer to the sealed data structure which was populated during by the sgx_seal_data function.

Return value

If the function succeeds, the number of bytes in the encrypted data buffer is returned. Othewise, the return value is `0xFFFFFFFF`. It is recommended that you check the return value before use 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	`sgx_tseal.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (simulation)

sgx_seal_data

This function is used to AES-GCM encrypt the input data. Two input data sets are provided: one is the data to be encrypted; the second is optional additional data that will not be encrypted but will be part of the GCM MAC calculation which also covers the data to be encrypted.

Syntax

```
sgx_status_t sgx_seal_data(
    const uint32_t additional_MACtext_length,
    const uint8_t * p_additional_MACtext,
    const uint32_t text2encrypt_length,
    const uint8_t * p_text2encrypt,
    const uint32_t sealed_data_size,
    sgx_sealed_data_t * p_sealed_data
);
```

Parameters

additional_MACtext_length [in]

Length of the additional Message Authentication Code (MAC) data in bytes. The additional data is optional and thus the length can be zero if no data is provided.

p_addtional_MACtext [in]

Pointer to the additional Message Authentication Code (MAC) data. This additional data is optional and no data is necessary (NULL pointer can be passed, but `additional_MACtext_length` must be zero in this case).

NOTE:

This data will not be encrypted. This data can be within or outside the enclave, but cannot cross the enclave boundary.

---

text2encrypt_length [in]

Length of the data stream to be encrypted in bytes. Must be non-zero.

p_text2encrypt [in]

Pointer to the data stream to be encrypted. Must not be NULL. Must be within the enclave.

sealed_data_size [in]

Number of bytes allocated for the sgx_sealed_data_t structure. The calling code should utilize helper function `sgx_calc_sealed_data_size` to determine the required buffer size.

p_sealed_data [out]

Pointer to the buffer to store the sealed data.

NOTE:

The calling code must allocate the memory for this buffer and should utilize helper function `sgx_calc_sealed_data_size` to determine the required buffer size. The sealed data must be within the enclave.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates an error if the parameters do not meet any of the following conditions:

- If `additional_MACtext_length` is non-zero, `p_additional_MACtext` cannot be NULL.
- `p_additional_MACtext` buffer can be within or outside the enclave, but cannot cross the enclave boundary.
- `p_text2encrypt` must be non-zero.
- `p_text2encrypt` buffer must be within the enclave.
- `sealed_data_size` must be equal to the required buffer size, which is calculated by the function `sgx_calc_sealed_data_size`.
- `p_sealed_data` buffer must be within the enclave.
- Input buffers cannot cross an enclave boundary.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

Indicates a crypto library failure or the RDRAND instruction fails to generate a random number.

Description

The `sgx_seal_data` function retrieves a key unique to the enclave and uses that key to encrypt the input data buffer. This function can be utilized to preserve secret data after the enclave is destroyed. The sealed data blob can be unsealed on future instantiations of the enclave.

The additional data buffer will not be encrypted but will be part of the MAC calculation that covers the encrypted data as well. This data may include information about the application, version, data, etc which can be utilized to identify the sealed data blob since it will remain plain text

Use `sgx_calc_sealed_data_size` to calculate the number of bytes to allocate for the sgx_sealed_data_t structure. The input sealed data buffer and text2encrypt buffers must be allocated within the enclave.

Requirements

Header	`sgx_tseal.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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_addtional_MACtext [in]

Pointer to the additional data to be MAC'ed of variable length. This additional data is optional and no data is necessary (NULL pointer can be passed, but `additional_MACtext_length` must be zero in this case).

NOTE:
This data will not be encrypted. This data can be within or outside the enclave, but cannot cross the enclave boundary.

---

text2encrypt_length [in]

Length of the data stream to be encrypted in bytes. Must be non-zero.

p_text2encrypt [in]

Pointer to the data stream to be encrypted of variable length. Must not be NULL. Must be within the enclave.

sealed_data_size [in]

Number of bytes allocated for `sealed_data_t` structure. The calling code should utilize helper function `sgx_calc_sealed_data_size` to determine the required buffer size.

p_sealed_data [out]

Pointer to the buffer that is populated by this function.

NOTE:
The calling code must allocate the memory for this buffer and should utilize helper function `sgx_calc_sealed_data_size` to determine the required buffer size. The sealed data must be within the enclave.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates an error if the parameters do not meet any of the following conditions:

- `If additional_MACtext_length` is non-zero, `p_additional_MACtext` cannot be NULL.

- p_additional_MACtext buffer can be within or outside the enclave, but cannot cross the enclave boundary.
- p_text2encrypt must be non-zero.
- p_text2encrypt buffer must be within the enclave.
- sealed_data_size must be equal to the required buffer size, which is calculated by the function sgx_calc_sealed_data_size.
- p_sealed_data buffer must be within the enclave.
- Input buffers cannot cross an enclave boundary.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

Indicates crypto library failure or the RDRAND instruction fails to generate a random number.

Description

The sgx_seal_data_ex is an extended version of sgx_seal_data. It provides parameters for you to identify how to derive the sealing key (key policy and attributes_mask). Typical callers of the seal library should be able to use sgx_seal_data and the default values provided for key_policy (MR_SIGNER) and an attribute mask which includes the RESERVED, INITED and DEBUG bits. Users of this function should have a clear understanding of the impact on using a policy and/or attribute_mask that is different from that in sgx_seal_data.

Requirement

Header	sgx_tseal.h
Library	libsgx_tservice.a or libsgx_tservice_sim.a (simulation)

sgx_unseal_data

This function is used to AES-GCM decrypt the input sealed data structure. Two output data sets result: one is the decrypted data; the second is the optional additional data that was part of the GCM MAC calculation but was not encrypted. This function provides the converse of sgx_seal_data and sgx_seal_data_ex.

Syntax

```
sgx_status_t sgx_unseal_data(
    const sgx_sealed_data_t * p_sealed_data,
    uint8_t * p_additional_MACtext,
    uint32_t * p_additional_MACtext_length,
    uint8_t * p_decrypted_text,
    uint32_t * p_decrypted_text_length
);
```

Parameters

p_sealed_data [in]

Pointer to the sealed data buffer to be AES-GCM decrypted. Must be within the enclave.

p_addtional_MACtext [out]

Pointer to the additional data part of the MAC calculation. This additional data is optional and no data is necessary. The calling code should call helper function `sgx_get_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 across the enclave boundary.
- `p_decrypted_text` and `p_decrypted_text_length` must be within the enclave.
- `p_decrypted_text` and `p_addtitional_MACtext` buffer must be big enough to receive the decrypted data.
- `p_sealed_data` buffer must be within the enclave.
- Input buffers cannot cross an enclave boundary.

SGX_ERROR_INVALID_CPUSVN

The CPUSVN in the sealed data blob is beyond the CPUSVN value of the platform.

SGX_ERROR_INVALID_ISVSVN

The ISVSVN in the sealed data blob is greater than the ISVSVN value of the enclave.

SGX_ERROR_MAC_MISMATCH

The tag verification failed during unsealing. The error may be caused by a platform update, software update, or sealed data blob corruption. This error is also reported if other corruption of the sealed data structure is detected.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

Indicates a cryptography library failure.

Description

The `sgx_unseal_data` function AES-GCM decrypts the sealed data so that the enclave data can be restored. This function can be utilized to restore secret data that was preserved after an earlier instantiation of this enclave saved this data.

The calling code needs to allocate the additional data buffer and the decrypted data buffer. To determine the minimum memory to allocate for these buffers, helper functions `sgx_get_add_mac_txt_len` and `sgx_get_encrypt_txt_len` are provided. The decrypted text buffer must be allocated within the enclave.

Requirements

Header	`sgx_tseal.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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_addtional_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	`sgx_tseal.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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=0xfffffffffffffff3, 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	libsgx_tservice.a or libsgx_tservice_sim.a (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_aadataorsgx_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_addtional_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_data`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	`sgx_tseal.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

sgx_sha256_init

`sgx_sha256_init` returns an allocated and initialized SHA algorithm context state. This should be part of the Init, Update ... Update, Final process when the SHA hash is to be performed over multiple datasets. If a complete

dataset is available, the recommend call is `sgx_sha256_msg` to perform the hash in a single call.

Syntax

```
sgx_status_t sgx_sha256_init(
    sgx_sha_state_handle_t* p_sha_handle
);
```

Parameters

p_sha_handle [out]

This is a handle to the context state used by the cryptography library to perform an iterative SHA256 hash. The algorithm stores the intermediate results of performing the hash calculation over data sets.

Return value

SGX_SUCCESS

The SHA256 state is allocated and initialized properly.

SGX_ERROR_INVALID_PARAMETER

The pointer `p_sha_handle` is invalid.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

The SHA256 state is not initialized properly due to an internal cryptography library failure.

Description

Calling `sgx_sha256_init` is the first set in performing a SHA256 hash over multiple datasets. The caller does not allocate memory for the SHA256 state that this function returns. The state is specific to the implementation of the cryptography library; thus the allocation is performed by the library itself. If the hash over the desired datasets is completed or any error occurs during the hash calculation process, `sgx_sha256_close` should be called to free the state allocated by this algorithm.

Requirements

Header	sgx_tcrypto.h
Library	libsgx_tcrypto.a

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	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	sgx_tcrypto.h
Library	libsgx_tcrypto.a

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 set 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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

sgx_rijndael128_cmac_msg

The `sgx_rijndael128_cmac_msg` function performs a standard 128bit CMAC hash over the input data buffer.

Syntax

```
sgx_status_t sgx_rijndael128_cmac_msg(
    const sgx_cmac_128bit_key_t *p_key,
    const uint8_t *p_src,
    uint32_t src_len,
    sgx_cmac_128bit_tag_t *p_mac
);
```

Parameters

p_key [in]

A pointer to key to be used in the CMAC hash operation. The size *must* be 128 bits.

p_src [in]

A pointer to the input data stream to be hashed. A zero length input buffer is supported, but the pointer must be non-NULL.

src_len [in]

Specifies the length on the input data stream to be hashed. A zero length input buffer is supported.

p_mac [out]

A pointer to the output 128-bit hash resulting from the CMAC calculation. This pointer must be non-NULL and the caller allocates memory for this buffer.

Return value

SGX_SUCCESS

The CMAC hash function is performed successfully.

SGX_ERROR_INVALID_PARAMETER

The key, source or MAC pointer is NULL.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation.

SGX_ERROR_UNEXPECTED

An unexpected internal cryptography library.

Description

The `sgx_rijndael128_cmac_msg` function performs a standard CMAC hash over the input data buffer. Only a 128-bit version of the CMAC hash is supported.

The function should be used if the complete input data stream is available. Otherwise, the Init, Update... Update, Final procedure should be used to compute a CMAC hash over multiple input data sets.

A zero-length input data buffer is supported, but the pointer must be non-NULL.

Requirements

Header	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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 set in performing a CMAC 128-bit hash over multiple datasets. The caller does not allocate memory for the CMAC state that this function returns. The state is specific to the implementation of the cryptography library and thus the allocation is performed by the library itself. If the hash over the desired datasets is completed or any error occurs during the hash calculation process, sgx_cmac128_close should be called to free the state allocated by this algorithm.

Requirements

Header	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	libsgx_tcrypto.a

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	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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: `pubKeyA = privKeyA * G`, where `G` is the base point of the elliptic curve.

2. Alice passes the public key to Bob.

3. Bob calculates his own public key pubKeyB by using his private key

privKeyB: `pubKeyB = privKeyB * G`, where `G` is a base point of the elliptic curve.

4. Bob passes the public key to Alice.

5. Alice gets Bob's public key and calculates the secret point shareKeyA. When calculating, she uses her own private key and Bob's public key and applies the following formula:

```
shareKeyA = privKeyA * pubKeyB = privKeyA * privKeyB *
G.
```

6. Bob gets Alice's public key and calculates the secret point shareKeyB. When calculating, he uses his own private key and Alice's public key and applies the following formula:

```
shareKeyB = privKeyB * pubKeyA = privKeyB * privKeyA *
G.
```

As the following equation is true `privKeyA * privKeyB * G = privKeyB * privKeyA * G`, the result of both calculations is the same,

that is, the equation shareKeyA = shareKeyB is true. The secret point serves as a secret key.

Shared secret shareKey is an x-coordinate of the secret point on the elliptic curve. The elliptic curve domain parameters must be hitherto defined by the function: `sgx_ecc256_open_context`.

Requirements

Header	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	libsgx_tcrypto.a

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	`libsgx_tcrypto.a`

sgx_rsa3072_verify

`sgx_rsa3072_verify` verifies the input digital signature for the given data-set 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	`libsgx_tcrypto.a`

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_dmq1,
    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_dmq1 [out]

Pointer to the RSA key factor `dmq1`.

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 speicified 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`, `dmq1`, `iqmp`).

Requirements

Header	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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 mod (p-1)`

p_rsa_key_dmq1 [in]

Pointer to the RSA factor `dmq1. dmq1=p mod (q-1)`

p_rsa_key_iqmp [in]

Pointer to the RSA factor `iqmp. iqmp=q^-1 mod 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`, `dmq1`, `iqmp`).

Requirements

Header	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	libsgx_tcrypto.a

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	`libsgx_tcrypto.a`

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	libsgx_tcrypto.a

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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]

Reserved for backward compatibility.

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	`libsgx_tkey_exchange.a`

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]

Reserved for backward compatibility.

derive_key_cb [in]

This a pointer to a call back routine matching the funtion 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	sgx_tkey_exchange.h sgx_tkey_exchange.edl
Library	libsgx_tkey_exchange.a

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 = AES-CMAC(key0, gab x-coordinate)
```

```
SGX_RA_KEY_MK = AES-CMAC(KDK,
0x01||'MK'||0x00||0x80||0x00)

SGX_RA_KEY_SK = 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	sgx_tkey_exchange.h sgx_tkey_exchange.edl
Library	libsgx_tkey_exchange.a

sgx_ra_close

Call the sgx_ra_close function to release the remote attestation and key exchange context after the process is done and the context isn't needed anymore.

Syntax

```
sgx_status_t sgx_ra_close(
    sgx_ra_context_t context
);
```

Parameters

context [in]

Context returned by sgx_ra_init.

Return value

SGX_SUCCESS

Indicates success.

SGX_ERROR_INVALID_PARAMETER

Indicates the context is invalid.

Description

At the end of a key exchange process, the caller needs to use this API in an enclave to clear and free memory associated with this remote attestation session.

Requirements

Header	sgx_tkey_exchange.h sgx_key_exchange.edl
Library	libsgx_tkey_exchange.a

sgx_dh_init_session

Initialize DH secure session according to the caller's role in the establishment.

Syntax

```
sgx_status_t sgx_dh_init_session(
    sgx_dh_session_role_t role,
    sgx_dh_session_t * session
);
```

Parameters

role [in]

Indicates which role the caller plays in the secure session establishment.

The value of role of the initiator of the session establishment must be `SGX_DH_SESSION_INITIATOR`.

The value of role of the responder of the session establishment must be `SGX_DH_SESSION_RESPONDER`.

session [out]

A pointer to the instance of the DH session which contains entire information about session establishment.

NOTE

The value of the pointer must be a valid address within an enclave, as well as the end address of the session structure.

Return value

SGX_SUCCESS

Session is initialized successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the input parameters is incorrect.

Requirements

Header	`sgx_dh.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (simulation)

.

sgx_dh_responder_gen_msg1

Generates MSG1 for the responder of DH secure session establishment and records ECC key pair in session structure.

Syntax

```
sgx_status_t sgx_dh_responder_gen_msg1(
    sgx_dh_msg1_t * msg1,
    sgx_dh_session_t * dh_session
);
```

Parameters

msg1 [out]

A pointer to an `sgx_dh_msg1_t` msg1 buffer. The buffer holding the msg1 message, which is referenced by this parameter, must be within the enclave.

The DH msg1 contains the responder's public key and report based target info.

dh_session [in/out]

A pointer that points to the instance of `sgx_dh_session_t`. The buffer holding the DH session information, which is referenced by this parameter, must be within the enclave.

NOTE

As output, the DH session structure contains the responder's public key and private key for the current session.

Return value

SGX_SUCCESS

MSG1 is generated successfully.

SGX_ERROR_INVALID_PARAMETER

Any of the input parameters is incorrect.

SGX_ERROR_INVALID_STATE

The API is invoked in incorrect order or state.

SGX_ERROR_OUT_OF_MEMORY

The enclave is out of memory.

SGX_ERROR_UNEXPECTED

An unexpected error occurred.

Requirements

Header	sgx_dh.h
Library	libsgx_tservice.a or libsgx_tservice_sim.a (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 thedh 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	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_dh.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_dh.h`
Library	`libsgx_tservice.a` or `libsgx_tservice_sim.a` (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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	`sgx_tprotected_fs.h sgx_tprotected_fs.edl`
Library	`libsgx_tprotected_fs.a`

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	libsgx_tcrypto.a

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 eror.

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	libsgx_tcrypto.a

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	libsgx_tcrypto.a

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 set 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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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 successfull.

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	`libsgx_tcrypto.a`

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 data-set 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	sgx_tcrypto.h
Library	libsgx_tcrypto.a

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

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	`sgx_tcrypto.h`
Library	`libsgx_tcrypto.a`

sgx_get_rsrv_mem_info

Get the start address and/or the max size info for the reserved memory area

Syntax

```
sgx_status_t sgx_get_rsrv_mem_info(
    void **addr, size_t* max_size
);
```

Parameters

addr [out]

The starting address of the reserved memory.

max_size [out]

The maximum size of the reserved memory.

Return value

SGX_SUCCESS

The reserved memory information is returned.

SGX_ERROR_INVALID_PARAMETER

One of the input parameters is invalid.

Description

`sgx_get_rsrv_mem_info` can be used to query the reserved memory area information, such as the starting address and the maximum size. The `addr` and `max_size` are not allowed to be NULL at the same time.

Requirements

Header	`sgx_rsrv_mem_mngr.h`
Library	`libsgx_tstdc.a`

sgx_alloc_rsrv_mem_ex

Allocate a range of EPC memory with a fixed address from the reserved memory area

Syntax

```
void* sgx_alloc_rsrv_mem_ex(
    void *desired_addr, size_t length
);
```

Parameters

desired_addr [in]

The desired starting address to allocate the reserved memory. Should be page aligned.

length [in]

The length of region to be allocated in bytes. Should be page aligned.

Return value

On success, sgx_alloc_rsrv_mem_ex returns a pointer with the desired starting address to the reserved memory region. Otherwise, returns NULL to indicate allocation failure.

Description

`sgx_alloc_rsrv_mem_ex` allocates a block of `length` bytes of EPC memory with a desired starting address from reserved memory area. The

desired_addr and length should be page aligned. Used when it needs to allocate a memory from a specific address from reserved memory area. If the starting address doesn't matter, please use sgx_alloc_rsrv_mem instead.

On success, a pointer to the memory is returned. On Intel(R) SGX 2.0 the initial permission for the allocated memory is RW, while on Intel(R) SGX 1.0 it depends on the ReservedMemExecutable flag in the Enclave Configuration File. If this flag is set, the allocated memory would be granted with RWX permission. Otherwise, the granted permission is RW.

After completion of the reserved memory operation, it needs to call sgx_free_rsrv_mem function to deallocate the memory resource.

Requirements

Header	sgx_rsrv_mem_mngr.h
Library	libsgx_tstdc.a

sgx_alloc_rsrv_mem

Allocate a range of EPC memory from the reserved memory area

Syntax

```
sgx_status_t sgx_alloc_rsrv_mem(
    size_t length
);
```

Parameters

length [in]

The memory length to be manipulated in bytes. Should be page aligned.

Return value

On success, sgx_alloc_rsrv_mem returns a pointer to the reserved memory region. Otherwise, returns NULL to indicate allocation failure.

Description

`sgx_alloc_rsrv_mem` allocates a block of `length` bytes of EPC memory from reserved memory area. The `length` should be page aligned. Used when the starting address doesn't matter.

On success, a memory pointer is returned. On Intel(R) SGX 2.0 the initial permission for the allocated memory is `RW`, while on Intel(R) SGX 1.0 it depends on the `ReservedMemExecutable` flag in the Enclave Configuration File. If this flag is set, the allocated memory would be granted with `RWX` permission. Otherwise, the granted permission is `RW`.

After completion of the reserved memory operation, it needs to call `sgx_free_rsrv_mem` function to deallocate the memory resource.

Requirements

Header	`sgx_rsrv_mem_mngr.h`
Library	`libsgx_tstdc.a`

sgx_free_rsrv_mem

Deallocate a range of EPC memory from the reserved memory area

Syntax

```
sgx_status_t sgx_free_rsrv_mem(
    void *addr, size_t length
);
```

Parameters

addr [in]

The starting address of region that would be freed. Should be page aligned.

length [in]

The length of the memory to be manipulated in bytes. Should be page aligned.

Return value

0

The EPC memory is freed successfully.

-1

The operation is failed.

Description

`sgx_free_rsrv_mem` is used to deallocate the memory from the reserved memory area allocated by the following functions:

sgx_alloc_rsrv_mem

sgx_alloc_rsrv_mem_ex

The input `addr` and `length` should be page aligned.

Requirements

Header	`sgx_rsrv_mem_mngr.h`
Library	`libsgx_tstdc.a`

sgx_tprotect_rsrv_mem

Modify the access permissions of the pages in the reserved memory area.

Syntax

```
sgx_status_t sgx_tprotect_rsrv_mem(
    void *addr, size_t length, int prot
);
```

Parameters

addr [in]

The starting address of region which needs to change access permission. Should be page aligned.

length [in]

The length of the memory to be manipulated in bytes. Should be page aligned.

prot [in]

The target memory protection.

Return value

SGX_SUCCESS

The permission is set to the target memory protection.

SGX_ERROR_INVALID_PARAMETER

Any of the input parameters is invalid.

SGX_ERROR_OUT_OF_MEMORY

Not enough memory to complete this operation.

SGX_ERROR_UNEXPECTED

Unexpected failure occurred.

Description

`sgx_tprotect_rsrv_mem` function sets the protection `prot` for the target reserved memory `[addr, addr+length-1]` similar to the system API `mprotect`. The `addr` and `length` should be page aligned.

`prot` is a combination of the following access flags: `SGX_PROT_NONE` or a bit-wise-or of the other values of the following list.

`SGX_PROT_NONE`	The memory cannot be accessed at all.
`SGX_PROT_READ`	The memory can be read.
`SGX_PROT_WRITE`	The memory can be modified.
`SGX_PROT_EXEC`	The memory can be executed.

NOTE:

This function is workable only on the Intel(R) SGX 2.0 enabled platform. For Intel(R) SGX 1.0 this function doesn't actually change any permission but perform a sanity check.

Requirements

Header	`sgx_rsrv_mem_mngr.h`
Library	`libsgx_tstdc.a`

tee_get_certificate_with_evidence

`tee_get_certificate_with_evidence` generates a self-signed X.509 certificate with embeded Intel® SGX ECDSA quote.

Syntax

```
quote3_error_t SGXAPI tee_get_certificate_with_evidence(
    const unsigned char *p_subject_name,
    const uint8_t *p_prv_key,
    size_t private_key_size,
    const uint8_t *p_pub_key,
    size_t public_key_size,
    uint8_t **pp_output_cert,
    size_t *p_output_cert_size
    );
```

Parameters

p_subject_name [in]

A string containing an X.509 distinguished name (DN)
for customizing the generated certificate. This name is also used
as the issuer name for this self-signed certificate See RFC5280 (https://tools.ietf.org/html/rfc5280) for details.

Example value "CN=Intel SGX Enclave,O=Intel Corporation,C=US"

p_prv_key [in]

A private key used to sign this certificate in PEM format

private_key_size [in]

The size of the private_key in bytes

p_pub_key [in]

A public key used as the certificate's subject key in PEM format

public_key_size [in]

The size of the public key in bytes

pp_output_cert [out]

A pointer to output certificate pointer

p_output_cert_size [out]

A pointer to the size of the output certificate above

Return value

SGX_QL_SUCCESS

All of the outputs are generated successfully.

SGX_QL_ERROR_INVALID_PARAMETER

Any of the parameters are invalid

SGX_QL_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation

SGX_QL_ATT_KEY_NOT_INITIALIZED

The platform quoting infrastructure does not have the attestation key available to generate quotes. sgx_qe_get_target_info() must be called again

SGX_QL_ATT_KEY_CERT_DATA_INVALID

The data returned by the platform library's sgx_ql_get_quote_config() is invalid

SGX_QL_OUT_OF_EPC

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

SGX_QL_ENCLAVE_LOST

Enclave lost after power transition or used in child process created by linux:-fork()

SGX_QL_ENCLAVE_LOAD_ERROR

Unable to load the enclaves required to initialize the attestation key. Could be due to file I/O error, loading infrastructure error or insufficient enclave memory

SGX_QL_ERROR_UNEXPECTED

An unexpected error was detected

Description

Calling `tee_get_certificate_with_evidence` generates a self-signed X.509 certificate with embedded Intel® SGX ECDSA quote. This API depends on Intel® SGX DCAP remote attestation.Review DCAP requirement before using this API.

Requirements

Header	`sgx_ttls.h, sgx_ttls.edl`
Library	`libsgx_ttls.a`

tee_free_certificate

`tee_free_evidence` frees the output X.509 certificate buffer which generate by API tee_get_certificate_with_evidence_in_enclave

Syntax

```
quote3_error_t SGXAPI tee_free_evidence(
    uint8_t* p_certificate
);
```

Parameters

p_certificate [in]

A pointer to output certificate buffer after called API `tee_get_certificate_with_evidence`

Return value

SGX_QL_SUCCESS

All of the outputs are generated successfully.

SGX_QL_ERROR_INVALID_PARAMETER

Any of the parameters are invalid.

SGX_QL_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation

SGX_QL_ERROR_UNEXPECTED

An unexpected error was detected

Requirements

Header	sgx_ttls.h, sgx_ttls.edl
Library	libsgx_ttls.a

tee_verify_certificate_with_evidence

`tee_verify_certificate_with_evidence` performs Intel® SGX quote and X.509 certificate verification inside an SGX enclave. The validation includes extracting quote extension from the certificate before validating the quote.

Syntax

```
quote3_error_t SGXAPI tee_verify_certificate_with_evid-
ence(
    const uint8_t *p_cert_in_der,
    size_t cert_in_der_len,
    const time_t expiration_check_date,
    sgx_ql_qv_result_t *p_qv_result,
    uint8_t **pp_supplemental_data,
    uint32_t *p_supplemental_data_size
    );
```

Parameters

p_cert_in_der [in]

A pointer to buffer holding certificate contents in DER format

in_der_len [in]

The size of certificate buffer above

expiration_check_date [in]

The date that verifier will use to determine if any of the verification collateral have expired

p_qv_result [in]

Quote verification result

pp_supplemental_data [out]

Optional input, a pointer to SGX quote verification supplemental data pointer

p_supplemental_data_size [out]

Optional input, the size of supplemental data above, only valid when you provide 'pp_supplemental_data'

Return value

SGX_QL_SUCCESS

Both X.509 certificate and quote verification passed,

But you can still refer to output parameters 'p_qv_result' for some non-critical errors, you can refer to 'p_qv_result' and supplemental data to define your own quote verification policy

SGX_QL_ERROR_INVALID_PARAMETER

Any of the parameters are invalid

SGX_QL_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation

SGX_QL_ERROR_UNEXPECTED

An unexpected error was detected.

Description

`tee_verify_certificate_with_evidence` extracts the quote from self-signed X.509 certificate, then calls Intel® ECDSA quote verification library and enclave to verify the quote. In addition, this API calls SgxSSL to verify the X.509 certificate.

Note that ifthis API returns success, it only means there was no critical error during quote verification. You should still refer to output 'p_qv_result' and 'p_supplemental_data' to check for warningssuch as verification collateral out of date warning. It is suggested to define your verification policy based the output or pass the output to relying party for further verification.

Requirements

Header	`sgx_ttls.h, sgx_ttls.edl`
Library	`libsgx_ttls.a`

tee_free_supplemental_data

`tee_free_supplemental_data` frees the quote verification supplemental data buffer, which is an output of the `tee_verify_certificate_with_evidence`API.

Note that this API should be called only when you askquote verification supplemental data in the `tee_verify_certificate_with_evidence`API.

Syntax

```
quote3_error_t SGXAPI tee_free_supplemental_data(
    uint8_t* p_supplemental_data
);
```

Parameters

p_supplemental_data [in]

A pointer to quote verification supplemental data, which is an output of the `tee_verify_certificate_with_evidence`API.

Return value

SGX_QL_SUCCESS

All of the outputs are generated successfully.

SGX_QL_ERROR_INVALID_PARAMETER

Any of the parameters are invalid.

SGX_QL_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation

SGX_QL_ERROR_UNEXPECTED

An unexpected error was detected

Requirements

Header	`sgx_ttls.h, sgx_ttls.edl`
Library	`libsgx_ttls.a`

tee_verify_certificate_with_evidence_host

`tee_verify_certificate_with_evidence_host` performs Intel® SGX quote and X.509 certificate verification in host side. The validation includes extracting quote extension from the certificate before validating the quote.

Note that the verification is not performed inside an SGX enclave. This API calls Intel® SGX QVL (Quote Verification Library) to verify quote, QvE (Quote Verification Enclave) is not involved. Before using this API, make sure the verification environment is secure.

Syntax

```
quote3_error_t SGXAPI tee_verify_certificate_with_evidence_host(
    const uint8_t *p_cert_in_der,
    size_t cert_in_der_len,
    const time_t expiration_check_date,
    sgx_ql_qv_result_t *p_qv_result,
    uint8_t **pp_supplemental_data,
    uint32_t *p_supplemental_data_size
    );
```

Parameters

p_cert_in_der [in]

A pointer to buffer holding certificate contents in DER format

in_der_len [in]

The size of certificate buffer above

expiration_check_date [in]

The date that verifier will use to determine if any of the verification collateral have expired

p_qv_result [in]

Quote verification result

pp_supplemental_data [out]

Optional input, a pointer to SGX quote verification supplemental data pointer

p_supplemental_data_size [out]

Optional input, the size of supplemental data above, only valid when you provide 'pp_supplemental_data'

Return value

SGX_QL_SUCCESS

Both X.509 certificate and quote verification passed,

But you can still refer to output parameters 'p_qv_result' for some non-critical errors, you can refer to 'p_qv_result' and supplemental data to define your own quote verification policy

SGX_QL_ERROR_INVALID_PARAMETER

Any of the parameters are invalid

SGX_QL_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation

SGX_QL_ERROR_UNEXPECTED

An unexpected error was detected.

Description

`tee_verify_certificate_with_evidence_host` extracts the quote from a self-signed X.509 certificate, then calls Intel® ECDSA quote verification library to verify the quote. In addition, this API calls OpenSSL to verify the X.509 certificate.

Note that if this API returns success, it only means there was no critical error during quote verification. You should still refer to output 'p_qv_result' and 'p_supplemental_data' to check for warnings such as verification collateral out of date warning. It is suggested to define your verification policy based the output or pass the output to relying party for further verification.

Requirements

Header	`sgx_utls.h`
Library	`libsgx_utls.a`

tee_free_supplemental_data_host

`tee_free_supplemental_data_host` frees the quote verification supplemental data buffer, which is an output of the `tee_verify_certificate_with_evidence_host`API.

Note that this API should be called only when you asked quote verification supplemental data in the `tee_verify_certificate_with_evidence_host`API.

Syntax

```
quote3_error_t SGXAPI tee_free_supplemental_data_host(
    uint8_t* p_supplemental_data
);
```

Parameters

p_supplemental_data [in]

A pointer to quote verification supplemental data, which is output of the `tee_verify_certificate_with_evidence_host`API.

Return value

SGX_QL_SUCCESS

All of the outputs are generated successfully.

SGX_QL_ERROR_INVALID_PARAMETER

Any of the parameters are invalid.

SGX_QL_ERROR_OUT_OF_MEMORY

Not enough memory is available to complete this operation

SGX_QL_ERROR_UNEXPECTED

An unexpected error was detected

Requirements

Header	sgx_utls.h
Library	libsgx_utls.a

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_rwlock_t
- sgx_thread_rwlockattr_t
- sgx_thread_cond_t
- sgx_thread_condattr_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_ra_context_t
- sgx_ra_key_128_t
- sgx_ra_key_type_t
- sgx_ra_msg1_t
- sgx_ra_msg2_t
- sgx_ra_msg3_t
- sgx_ecall_get_ga_trusted_t

- sgx_ecall_get_msg3_trusted_t
- sgx_ecall_proc_msg2_trusted_t
- sgx_platform_info_t
- sgx_update_info_bit_t
- sgx_dh_msg1_t
- sgx_dh_msg2_t
- sgx_dh_msg3_t
- sgx_dh_msg3_body_t
- sgx_dh_session_enclave_identity_t
- sgx_dh_session_role_t
- sgx_dh_session_t
- 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	sgx_eid.h

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	sgx_error.h

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 {
    uint64_t processed;
    uint64_t missed;
} sgx_uswitchless_worker_stats_t;
```

Members

processed

64-bit counter that counts the number of tasks that all workers have processed.

missed

64-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_DEFUALT_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	`sgx_uswitchless.h`

sgx_exception_vector_t

The sgx_exception_vector_t enumeration contains the enclave supported exception vectors. If the exception vector is #BP, the exception type is SGX_EXCEPTION_SOFTWARE; otherwise, the exception type is SGX_EXCEPTION_HARDWARE.

Syntax

```
typedef enum _sgx_exception_vector_t
{
    SGX_EXCEPTION_VECTOR_DE = 0, /* DIV and DIV instructions */
    SGX_EXCEPTION_VECTOR_DB = 1, /* For Intel use only */
    SGX_EXCEPTION_VECTOR_BP = 3, /* INT 3 instruction */
    SGX_EXCEPTION_VECTOR_BR = 5, /* BOUND instruction */
    SGX_EXCEPTION_VECTOR_UD = 6, /* UD2 instruction or reserved
    opcode */
    SGX_EXCEPTION_VECTOR_MF = 16, /* x87 FPU floating-point or
    WAIT/FWAI instruction. */
    SGX_EXCEPTION_VECTOR_AC = 17, /* Any data reference in memory */
    SGX_EXCEPTION_VECTOR_XM = 19, /* SSE/SSE2/SSE3 instruction */
} sgx_exception_vector_t;
```

Requirements

Header	sgx_trts_exception.h

sgx_exception_type_t

The sgx_exception_type_t enumeration contains values that specify the exception type. If the exception vector is #BP (BreakPoint), the exception type is SGX_EXCEPTION_SOFTWARE; otherwise, the exception type is SGX_EXCEPTION_HARDWARE.

Syntax

```
typedef enum _sgx_exception_type_t
{
    SGX_EXCEPTION_HARDWARE = 3,
    SGX_EXCEPTION_SOFTWARE = 6,
} sgx_exception_type_t;
```

Requirements

Header	sgx_trts_exception.h

sgx_cpu_context_t

The `sgx_cpu_content_t` structure contains processor-specific register data. Custom exception handling uses `sgx_cpu_context_t` structure to record the CPU context at exception time.

Syntax

```
#if defined (_M_X64) || defined (__x86_64__)
    typedef struct _cpu_context_t
    {
        uint64_t rax;
        uint64_t rcx;
        uint64_t rdx;
        uint64_t rbx;
        uint64_t rsp;
        uint64_t rbp;
        uint64_t rsi;
        uint64_t rdi;
        uint64_t r8;
        uint64_t r9;
        uint64_t r10;
        uint64_t r11;
        uint64_t r12;
        uint64_t r13;
        uint64_t r14;
        uint64_t r15;
        uint64_t rflags;
        uint64_t rip;
    } sgx_cpu_context_t;
#else
    typedef struct _cpu_context_t
    {
        uint32_t eax;
        uint32_t ecx;
        uint32_t edx;
        uint32_t ebx;
        uint32_t esp;
        uint32_t ebp;
        uint32_t esi;
        uint32_t edi;
        uint32_t eflags;
        uint32_t eip;
    } sgx_cpu_context_t;
#endif
```

Members

rax, rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8 – r15

64-bit general purpose registers

rflags

64-bit program status and control register

rip

64-bit instruction pointer

eax, ecx, edx, ebx, esp, ebp, esi, edi

32-bit general purpose registers

eflags

32-bit program status and control register

eip

32-bit instruction pointer

Requirements

Header	sgx_trts_exception.h

sgx_exception_info_t

A structure of this type contains an exception record with a description of the exception and processor context record at the time of exception.

Syntax

```
typedef struct _exception_info_t
{
    sgx_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	`sgx_spinlock.h`

sgx_thread_t

Data type to uniquely identify a trusted thread.

Syntax

```
typedef uintptr * sgx_thread_t;
```

Members

`sgx_thread_t` is an opaque data type with no member fields visible to users. This data type is subject to change. Thus, enclave code should not rely on the contents of this data object.

Requirements

Header	`sgx_thread.h`

sgx_thread_mutex_t

Data type for a trusted mutex object.

Syntax

```
typedef struct sgx_thread_mutex
{
    size_t m_refcount;
    uint32_t m_control;
    volatile uint32_t m_lock;
    sgx_thread_t m_owner;
    sgx_thread_queue_t m_queue;
} sgx_thread_mutex_t;
```

Members

m_control

Flags to define whether a mutex is recursive or not.

m_refcount

Reference counter of the mutex object. It will be increased by 1 if the mutex is successfully acquired, and be decreased by 1 if the mutex is released.

NOTE

The counter will be greater than one only if the mutex is recursive.

m_lock

The spin lock used to guarantee atomic updates to the mutex object.

m_owner

The thread that currently owns the mutex writes its unique thread identifier in this field, which otherwise is NULL. This field is used for error checking, for instance to ensure that only the owner of a mutex releases it.

m_queue

Ordered list of threads waiting to acquire the ownership of the mutex. The queue itself is a structure containing a head and a tail for quick insertion and removal under FIFO semantics.

Requirements

Header	sgx_thread.h

sgx_thread_mutexattr_t

Attribute for the trusted mutex object.

Syntax

```
typedef struct sgx_thread_mutex_attr
{
    unsigned char m_dummy;
} sgx_thread_mutexattr_t;
```

Members

m_dummy

Dummy member not supposed to be used.

Requirements

Header	sgx_thread.h

sgx_thread_rwlock_t

Data type for a trusted mutex object.

Syntax

```
typedef struct _sgx_thread_rwlock_t
{
    uint32_t m_reader_count;
    uint32_t m_writers_waiting;
    volatile uint32_t m_lock;
    sgx_thread_t m_owner;
    sgx_thread_queue_t m_reader_queue;
    sgx_thread_queue_t m_writer_queue;
} sgx_thread_rwlock_t;
```

Members

m_reader_count

The number of readers currently holding the lock. It is increased by 1 whenever a reader lock is successfully acquired, and is decreased by 1 when a reader lock is released.

m_writers_waiting

The number of threads waiting on a writer lock.

m_lock

The spin lock used to guarantee atomic updates to the rwlock object.

m_owner

The thread that currently owns the writer lock, writing 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 writer lock releases it.

m_reader_queue

Ordered list of threads waiting to acquire the reader lock. The queue itself is a structure containing a head and a tail for quick insertion and removal under FIFO semantics.

m_writer_queue

Ordered list of threads waiting to acquire the writer lock. 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_rwlockattr_t

Attribute for the trusted rwlock object.

Syntax

```
typedef struct _sgx_thread_rwlock_attr_t
{
    unsigned char m_dummy;
} sgx_thread_rwlockattr_t;
```

Members

m_dummy

Dummy member not supposed to be used.

Requirements

Header	sgx_thread.h

sgx_thread_cond_t

Data type for a trusted condition variable.

Syntax

```
typedef struct sgx_thread_cond
{
    sgx_spinlock_t m_lock;
    sgx_thread_queue_t m_queue;
} sgx_thread_cond_t;
```

Members

m_lock

The spin lock used to guarantee atomic updates to the condition variable.

m_queue

Ordered list of threads waiting on the condition variable. The queue itself is a structure containing a head and a tail for quick insertion and removal under FIFO semantics.

Requirements

Header	sgx_thread.h

sgx_thread_condattr_t

Attribute for the trusted condition variable.

Syntax

```
typedef struct sgx_thread_cond_attr
{
    unsigned char m_dummy;
} sgx_thread_condattr_t;
```

Members

m_dummy

Dummy member not supposed to be used.

Requirements

Header	sgx_thread.h

sgx_misc_select_t

Enclave misc select bits. The value is 4 byte in length. Currently all the bits are reserved for future extension.

Requirements

Header	sgx_attributes.h

sgx_attributes_t

Enclave attributes definition structure.

NOTE

When specifying an attributes mask used in key derivation, at a minimum the flags that should be set are INITED, DEBUG and RESERVED bits.

NOTE

The XGETBV instruction can be executed to determine the register sets, which are 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 the used to store a derived key from for example the `sgx_get_key` function.

Requirements

Header	sgx_key.h

sgx_key_request_t

Data structure of 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_PROVISION	0x0001	Provisioning key
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.

NOTE

If MRSIGNER is used, another enclave with the same ISV_SVN could derive the key as well. This option is useful for applications that instantiate more than one enclave and want to pass data. The derived key could be used in the encryption process for the data passed between the enclaves.

isv_svn

The ISV security version number that should be used in the key derivation.

reserved1

Reserved for future use. Must be zero.

cpu_svn

The TCB security version number that should be used in the key derivation.

attribute_mask

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	`sgx_key.h`

sgx_measurement_t

`sgx_measurement_t` is a 256-bit value representing the enclave measurement.

Syntax

```
#define SGX_HASH_SIZE 32

typedef struct _sgx_measurement_t {
    uint8_t m[SGX_HASH_SIZE];
} sgx_measurement_t;
```

Requirements

Header	`sgx_report.h`

sgx_mac_t

This type is utilized as storage for the 128-bit CMAC value of the report data.

Requirements

Header	`sgx_report.h`

sgx_report_data_t

`sgx_report_data_t` is a 512-bit value used for communication between the enclave and the target enclave. This is one of the inputs to the `sgx_create_report` function.

Syntax

```
#define SGX_REPORT_DATA_SIZE 64

typedef struct _sgx_report_data_t {
    uint8_t d[SGX_REPORT_DATA_SIZE];
} sgx_report_data_t;
```

Requirements

Header	`sgx_report.h`

sgx_prod_id_t

A 16-bit value representing the ISV enclave product ID. This value is used in the derivation of some keys.

Requirements

Header	sgx_report.h

sgx_target_info_t

Data structure of report target information. This is an input to 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 _targe_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 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.

xeid

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_ql_att_key_id_t

Aa single attestation key. Contains both QE identity and the attestation algorithm ID.

Syntax

```
typedef struct _sgx_ql_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_ql_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_ql_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	sgx_quote.h

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_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 (Litte Endian)

Key derivation function id.

sign_gb_ga (Litte Endian)

ECDSA Signature of (`g_b||g_a`), using the service provider's ECDSA private key corresponding to the public key specified in `sgx_ra_init` 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 VKcalculation 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	`sgx_key_exchange.h`

sgx_ecall_get_ga_trusted_t

Function pointer of proxy function generated from `sgx_tkey_exchange.edl`.

Syntax

```
typedef sgx_status_t (* sgx_ecall_get_ga_trusted_t)(
    sgx_enclave_id_t eid,
    int* retval,
    sgx_ra_context_t context,
    sgx_ec256_public_t *g_a // Little Endian
);
```

Note that the 4th parameter this function takes should be in little endian format.

Requirements

Header	`sgx_ukey_exchange.h`

sgx_ecall_proc_msg2_trusted_t

Function pointer of proxy function generated from `sgx_tkey_exchange.edl`.

Syntax

```
typedef sgx_status_t (* sgx_ecall_proc_msg2_trusted_t)(
    sgx_enclave_id_t eid,
    int* retval,
    sgx_ra_context_t context,
    const sgx_ra_msg2_t *p_msg2,
    const sgx_target_info_t *p_qe_target,
    sgx_report_t *p_report,
    sgx_quote_nonce_t *p_nonce
);
```

Requirements

Header	`sgx_ukey_exchange.h`

sgx_ecall_get_msg3_trusted_t

Function pointer of proxy function generated from `sgx_tkey_exchange.edl`.

Syntax

```
typedef sgx_status_t (* sgx_ecall_get_msg3_trusted_t)(
    sgx_enclave_id_t eid,
    int* retval,
    sgx_ra_context_t context,
    uint32_t quote_size,
    sgx_report_t* qe_report,
    sgx_ra_msg3_t *p_msg3,
    uint32_t msg3_size
);
```

Requirements

Header	sgx_ukey_exchange.h

sgx_platform_info_t

This opaque data structure indicates the platform information received from Intel Attestation Server.

Syntax

```
#define SGX_PLATFORM_INFO_SIZE 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_config_svn_t

16-bits value representing the enclave CONFIGSVN. This value is used in the derivation of some keys.

Requirements

Header	sgx_key.h

sgx_config_id_t

64-bytes value representing the enclave CONFIGID. This value is used in the derivation of some keys.

Requirements

Header	sgx_key.h

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	sgx_report.h

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	sgx_report.h

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 18 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 or used in child process created by fork().
0x0005	SGX_ERROR_ INVALID_STATE	The API is invoked in incorrect order or state.

0x0008	SGX_ERROR_ FEATURE_NOT_ SUPPORTED	The feature is not supported.
0x000a	SGX_ERROR_ MEMORY_MAP_ FAILURE	Failed to reserve memory for the enclave.
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: • ECALL is not public. • ECALL is blocked by the dynamic entry table. • A nested ECALL is not allowed during global initialization.
0x1008	SGX_ERROR_ OCALL_NOT_ ALLOWED	OCALL is not allowed during exception handling.
0x1009	SGX_ERROR_ STACK_OVERRUN	Stack overrun occurs within the enclave.
0x2000	SGX_ERROR_ UNDEFINED_ SYMBOL	The enclave contains an undefined symbol.
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_ CONFLICT	Page mapping failed in the driver.
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 `sgx_sign` or the uRTS is incompatible with the current platform.
0x200E	SGX_ERROR_ MODE_ INCOMPATIBLE	The target enclave (32/64 bit or HS/Sim) mode is incompatible with the uRTS mode.
0x200F	SGX_ERROR_ ENCLAVE_FILE_ ACCESS	Cannot open the enclave file.
0x2010	SGX_ERROR_ INVALID_MISC	The MiscSelect or MiscMask settings are incorrect.
0x2011	SGX_ERROR_ INVALID_LAUNCH_ TOKEN	The launch token is incorrect.
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_ SERVICE_INVALID_ PRIVILEDGE	Enclave has no privilege to get a launch token.
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_	Intel® EPID Provisioning failed because the platform is not recognized by the back-end server.

	PLATFORM	
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
0x6003	SGX_ERROR_PCL_MAC_MISMATCH	(Intel® SGX PCL) The runtime AES-GCM-128 MAC result of an encrypted section does not match the one at build time.
0x6004	SGX_ERROR_PCL_SHA_MISMATCH	(Intel® SGX PCL) The runtime SHA256 of the decryption key does not match the one at build time.
0x6005	SGX_ERROR_PCL_GUID_MISMATCH	(Intel® SGX PCL) The GUID in the decryption key sealed blob does not match the one at build time.
0x7001	SGX_ERROR_FILE_BAD_STATUS	The file is in a bad status. Run `sgx_clearerr` 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_ ATT_KEY_ID	Platform quoting infrastructure does not support the key.
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 `sgx_get_quote_ config()` of the platform library is invalid.
0x8005	SGX_ERROR_ PLATFORM_CERT_ UNAVAILABLE	The PCK Cert for the platform is not available.

Appendix

This topic provides the following reference information:

Unsupported GCC* Compiler Options for Enclaves

The following GCC* options are not supported to build enclaves.

Table 19 Unsupported GCC Compiler Options

Option	Category	Remark
-fgnu-tm	Options controlling C dialect.	Depends on libitm (transactional memory).
-fhosted		OS functions not supported within enclaves.
-fuse-cxa-atexit	Options controlling C++ dialect.	Depends on atexit(), which is not supported within an enclave.
All options	Options controlling objective-C and objective-C++.	Objective C/C++ not supported.
All options	Options for debugging a program.	All options because of runtime support required. Separate Intel® SGX debugger support provided.
-fmudflap, -fmudflapth, –fmudflapir	Optimization options.	Dependent on libmudflap.
-fexec-charset=charset, -fwide-exec-charset=charset	Options controlling the preprocessor.	Only providing partial support for UTF-8.
-x objective-c		Objective-C is not supported within an enclave.

-lobjc	Linker options.	Objective C not supported.
-pie, -s		Used for executables.
-shared-libgcc, -static-libgcc		Enclaves cannot depend on libgcc.
-static-libstdc++		Intel® SGX SDK provides an Intel SGX version of the C++ standard library.
-T script		Need to control the format of enclave code.
-mglibc	Hardware models and configurations for GNU*/Linux* options.	Intel SGX SDK provides an Intel SGX compatible C standard library.
-muclibc, -mbionic, -mandroid, -tno-android-cc, -tno-android-ld		Not applicable.
-msoft-float	Hardware models and configurations for Intel & AMD* x86-x64 options.	Run-time emulation of floating point is not supported.
-m96bit-long-double		96-bit not supported.
-mthreads		Depends on mingwthrd.
-mcmodel=small, -mcmodel=kernel, -mcmodel=medium, -mcmodel=large		Linker will fail.
All options	Hardware models and configurations for Intel & AMD* x86-x64 Windows* options	All options because these are only used with Cygwin* or MinGW*.
-fbounds-check	Options for code generation conventions	Currently for Java* and Fortran* front-ends, not C/C++.
-fpie, -fPIE		Only pertains to executable files.
-fpie, -pie		compilation option -fpie and linking option -pie cannot be used at the same time under simulation mode if TLS support is required.
-fpie, -shared -fpic		compilation option -fpie and linking option -shared -fpic cannot be used at the same time under both simulation mode and 64-bit hardware mode if TLS support is required.
-finstrument-functions		ISV would need to provide support for functions __

		`_cyg_profile_func_enter` and `__cyg_profile_func_exit` if this option is needed.
-fsplit-stack		Requires libgcc runtime support.

Unsupported GCC* Built-in Functions

The following table illustrates unsupported GCC* built-in functions inside the enclave. Using any of these built-in functions will result in a linker error during the compilation of the enclave.

The complete list of GCC built-in functions is available at http://gcc.gnu.org/onlinedocs/gcc-4.7.2/gcc/X86-Built_002din-Functions.html#X86-Built_002din-Functions.

Table 20 Unsupported GCC Compiler Built-in Functions

Non supported: Math built-ins		
__builtin_signbitd32	__builtin_signbitd64	__builtin_signbitd128
__builtin_finited32	__builtin_finited64	__builtin_finited128
__builtin_isinfd32	__builtin_isinfd64	__builtin_isinfd128
__builtin_isnand32	__builtin_isnand64	__builtin_isnand64
Not Supported: String/memory built-ins		
__builtin_strcat	__builtin_strcpy	__builtin_strdup
__builtin_stpcpy		
Not Supported: I/O related built-ins		
__builtin_fprintf	__builtin_fprintf_unlocked	__builtin_putc
__builtin_putc_unlocked	__builtin_fputc	__builtin_fputc_unlocked
__builtin_fputs	__builtin_fputs_unlocked	__builtin_fscanf
__builtin_fwrite	__builtin_fwrite_unlocked	__builtin_printf
__builtin_printf_unlocked	__builtin_putchar	__builtin_putchar_unlocked
__builtin_puts	__builtin_puts_unlocked	__builtin_scanf
__builtin_sprintf	__builtin_sscanf	__builtin_vfprintf
__builtin_vfscanf	__builtin_vprintf	__builtin_vscanf
__builtin_vsprintf	__builtin_vsscanf	
Not Supported: wctype built-in		
__builtin_iswalnum	__builtin_iswalpha	__builtin_iswblank
__builtin_iswcntrl	__builtin_iswdigit	__builtin_iswgraph
__builtin_iswlower	__builtin_iswprint	__builtin_iswpunct
__builtin_iswspace	__builtin_iswupper	__builtin_iswxdigit

__builtin_towlower	__builtin_towupper	
Not Supported: Process control built-ins		
__builtin_execl	__builtin_execlp	__builtin_execle
__builtin_execv	__builtin_execvp	__builtin_execve
__builtin_exit	__builtin_fork	__builtin__exit
__builtin__Exit		
Non Supported: Object size checking built-ins		
__builtin___fprintf_chk	__builtin___printf_chk	__builtin___vfprintf_chk
__builtin___vprintf_chk		
Non Supported: Miscellaneous built-ins		
__builtin_dcgettext	__builtin_dgettext	__builtin_gettext
__builtin_strfmon		
Non Supported: Profiling Hooks		
__cyg_profile_func_enter	__cyg_profile_func_exit	
Non Supported: TLS Emulation		
target.emutls.get_address	target.emutls.register_common	
Non supported: Ring 0 built-ins		
_writefsbase_u32	_writefsbase_u64	_writegsbase_u32
_writegsbase_u64	__rdpmc	__rdtsc
__rdtscp		

Non Supported: OpenMP* built-ins	
__builtin_omp_get_thread_num	__builtin_omp_get_num_threads
__builtin_GOMP_atomic_start	__builtin_GOMP_atomic_end
__builtin_GOMP_barrier	__builtin_GOMP_taskwait
__builtin_GOMP_taskyield	__builtin_GOMP_critical_start
__builtin_GOMP_critical_end	__builtin_GOMP_critical_name_start
__builtin_GOMP_critical_name_end	__builtin_GOMP_loop_static_start
__builtin_GOMP_loop_dynamic_start	__builtin_GOMP_loop_guided_start
__builtin_GOMP_loop_runtime_start	__builtin_GOMP_loop_ordered_static_start
__builtin_GOMP_loop_ordered_dynamic_start	__builtin_GOMP_loop_ordered_guided_ start
__builtin_GOMP_loop_ordered_runtime_start	__builtin_GOMP_loop_static_next
__builtin_GOMP_loop_dynamic_next	__builtin_GOMP_loop_guided_next
__builtin_GOMP_loop_runtime_next	__builtin_GOMP_loop_ordered_static_next
__builtin_GOMP_loop_ordered_dynamic_next	__builtin_GOMP_loop_ordered_guided_next
__builtin_GOMP_loop_ordered_runtime_next	__builtin_GOMP_loop_ull_static_start
__builtin_GOMP_loop_ull_dynamic_start	__builtin_GOMP_loop_ull_guided_start
__builtin_GOMP_loop_ull_runtime_start	__builtin_GOMP_loop_ull_ordered_static_

	start
__builtin_GOMP_loop_ull_ordered_dynamic_start	__builtin_GOMP_loop_ull_static_next
__builtin_GOMP_loop_ull_ordered_guided_start	__builtin_GOMP_loop_ull_dynamic_next
__builtin_GOMP_loop_ull_ordered_runtime_start	__builtin_GOMP_loop_ull_guided_next
__builtin_GOMP_loop_ull_runtime_next	__builtin_GOMP_loop_ull_ordered_static_next
__builtin_GOMP_loop_ull_ordered_dynamic_next	__builtin_GOMP_parallel_loop_static_start
__builtin_GOMP_loop_ull_ordered_guided_next	__builtin_GOMP_parallel_loop_dynamic_start
__builtin_GOMP_loop_ull_ordered_runtime_next	__builtin_GOMP_parallel_loop_guided_start
__builtin_GOMP_parallel_loop_runtime_start	__builtin_GOMP_loop_end
__builtin_GOMP_loop_end_nowait	__builtin_GOMP_ordered_start
__builtin_GOMP_ordered_end	__builtin_GOMP_parallel_start
__builtin_GOMP_parallel_end	__builtin_GOMP_task
__builtin_GOMP_sections_start	__builtin_GOMP_sections_next
__builtin_GOMP_parallel_sections_start	__builtin_GOMP_sections_end
__builtin_GOMP_sections_end_nowait	__builtin_GOMP_single_start
__builtin_GOMP_single_copy_start	__builtin_GOMP_single_copy_end

Unsupported C Standard Functions

You cannot use the following Standard C functions within the enclave; otherwise, the compilation would fail.

Table 21 Unsupported C Standard Functions

Header file	Header file in Intel SGX?	Unsupported Definition	
		Macros/Types	Functions
complex.h	No	complex, _complex_I, imaginary, _imaginary_I, I,	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(),

Header file	Header file in Intel SGX?	Unsupported Definition	
		Macros/Types	**Functions**
		#pragma STDC CX_LIMITED_RANGE on-off-switch	ctanh(), ctanhf(), ctanhl(), cexp(), cexpf(), cexpl(), clog(), clogf(), clogl(), cabs(), cabsf(), cabsl(), cpow(), cpowf(), cpowl(), csqrt(), csqrtf(), csqrtl(), carg(), cargf(), cargl(), cimag(), cimagf(), cimagl(), conj(), conjf(), conjl(), cproj(), cprojf(), cprojl(), creal(), crealf(), creall()
fenv.h	No	fenv_t, fexcept_t, FE_DIVBYZERO, FE_INEXACT, FE_INVALID, FE_OVERFLOW, FE_UNDERFLOW, FE_ALL_EXCEPT, FE_DOWNWARD, FE_TONEAREST, FE_TOWARDZERO, FE_UPWARD, FE_DFL_ENV, #pragma STDC FENV_ACCESS on-off-switch	feclearexcept(), fegetexceptflag(), feraiseexcept(), fesetexceptflag(), fetestexcept(), fegetround(), fesetround(), fegetenv(), feholdexcept(), fesetenv(), feupdateenv()
inttypes.h	Yes	SCNdN, SCNiN, SCNoN, SCNuN, SCNxN, SCNdLEASTN, SCNiLEASTN, SCNoLEASTN, SCNuLEASTN, SCNxLEASTN, SCNdFASTN, SCNiFASTN, SCNoFASTN,	wcstoimax(), wcstoumax()

Header file	Header file in Intel SGX?	Unsupported Definition	
		Macros/Types	**Functions**
		SCNuFASTN, SCNxFASTN, SCNdMAX, SCNiMAX, SCNoMAX, SCNuMAX, SCNxMAX, SCNdPTR, SCNiPTR, SCNoPTR, SCNuPTR, SCNxPTR	
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()
stdio.h	Yes	fpos_t, _IOFBF, _IOLBF, _IONBF, FILENAME_MAX, FOPEN_MAX, L_tmpnam, SEEK_CUR, SEEK_END, SEEK_SET, TMP_MAX, stderr, stdin, stdout	remove(), rename(), tmpfile(), tmpnam(), fclose(), fflush(), fopen(), freopen(), setbuf(), setvbuf(), fprintf(), fscanf(), printf(), scanf(), sprintf(), sscanf(), vfprintf(), vfscanf(), vprintf(), vscanf(), vsprintf(), vsscanf(), fgetc(), fgets(), fputc(), fputs(), getc(), getchar(), gets(), putc(), putchar(), puts(), ungetc(), fread(), fwrite(), fgetpos(), fseek(), fsetpos(), ftell(), rewind(), clearerr(), feof(), ferror(), perror()
stdlib.h	Yes		rand(), srand(), exit(), _Exit(), getenv(), system()
string.h	Yes		strcpy(), strcat(), strstr()*

Header file	Header file in Intel SGX?	Unsupported Definition	
		Macros/Types	Functions
tgmath.h	No		
time.h	Yes		clock(), mktime(), time(), ctime(), gmtime(), localtime()
wchar.h	Yes		fwprintf(), fwscanf(), swscanf(), vfwprintf(), vfwscanf(), vswscanf(), vwprintf(), vwscanf(), wprintf(), wscanf(), fgetwc(), fgetws(), fputwc(), fputws(), fwide(), getwc(), getwchar(), putwc(), putwchar(), ungetwc(), wcstod(), wcstof(), wcstold(), wcstol(), wcstoll(), wcstoul(), wcstoull(), wcscpy(), wcscat(), wcsftime()
wctype.h	Yes		iswalnum(), iswalpha(), iswblank(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(), wctype(), towlower(), towupper(), towctrans(), wctrans(),

(*) The trusted standard C library does not support `char strstr(const char*, const char*)`. However, it does support the variant `const char* strstr (const char*, const char*)` is supported.

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.

Unsupported C++ Standard Classes and Functions

The following table lists unsupported C++11 classes and functions inside the enclave.

Table 22 Utilities library

Header File	Support	Unsupported Classes
cstdlib	Partial	
csignal	No	
csetjmp	No	

cstdarg	Yes	
typeinfo	Partial	
typeindex	Yes	
type_traits	Yes	
bitset	Yes	
functional	Yes	
utility	Yes	
ctime	Partial	
chrono	No	
cstddef	Yes	
initializer_list	Yes	
tuple	Yes	

Table 23 Dynamic memory management

Header File	Support	Unsupported Classes
memory	Yes	
new	Partial	
scoped_allocator	Yes	

Table 24 Numeric limits

Header File	Support	Unsupported Classes
cfloat	Yes	
cinttypes	Partial	
climits	Yes	
cstdint	Partial	
limits	Yes	

Table 25 Error handling

Header File	Support	Unsupported Classes
cassert	Yes	
cerrno	Yes	
exception	Partial	`nested_exception`
stdexcept	Yes	
system_error	Yes	

Table 26 Strings library

Header File	Support	Unsupported Classes
cctype	Partial	
cstring	Partial	
cuchar	No	
cwchar	Partial	
cwctype	Partial	
string	Yes	

Table 27 Containers library

Header File	Support	Unsupported Classes
array	Yes	
dequeue	Yes	
forward_list	Yes	
list	Yes	
map	Yes	
queue	Yes	
set	Yes	
stack	Yes	
unordered_map	Yes	
unordered_set	Yes	
vector	Yes	

Table 28 Algorithms library

Header File	Support	Unsupported Classes
algorithm	Partial	

Table 29 Iterators library

Header File	Support	Unsupported Classes
iterator	Yes	

Table 30 Numerics library

Header File	Support	Unsupported Classes
cfenv	No	
cmath	Partial	
complex	Yes	
numeric	Yes	

random	No	
ratio	Yes	
valarray	Yes	

Table 31 Input/Output library

Header File	Support	Unsupported Classes
cstdio	Partial	
fstream	No	
iomanip	Partial	`Functions get_money, put_money, get_time, put_time are not supported.`
ios	Yes	
iosfwd	Yes	
iostream	No	
istream	Partial	`Formatted input is not supported.`
ostream	Partial	`Formatted output is not supported.`
sstream	Partial	
streambuf	Yes	
strstream	Yes	

Table 32 Localization library

Header File	Support	Unsupported Classes
clocale	No	
codecvt	No	
locale	Partial	`std::ctype<char> and std::ctype<wchar_t> are the only supported facets.`

Table 33 Regular expressions library

Header File	Support	Unsupported Classes
regex	No	

Table 34 Atomic operations library

Header File	Support	Unsupported Classes
atomic	Yes	

Table 35 Thread library

Header File	Support	Unsupported Classes
condition_ variable	Partial	
future	No	
mutex	Partial	`timed_mutex, recursive_timed_mutex`
thread	No	

Table 36 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 37 Unsupported C and C++ Keywords

__transaction_atomic	__transaction_relaxed	__transaction_cancel

The following GCC* specific attributes are not supported in an enclave.

Table 38 Unsupported GCC* Compiler Attributes

destructor	transaction_callable	transaction_unsafe
transaction_safe	transaction_may_cancel_outer	transaction_pure
transaction_wrap	disinterrupt	

C++11 Support on Linux* OS

Although C++11 is considered the baseline, the availability of certain C++11 features depends on the GCC* compiler version and/or a specific compiler option. The table below summarizes the C++11 features available when you use the GCC compiler option -std=c++11 and link the enclave with `sgx_tcxx` as the standard C++ library.

Table 39 Supported C++11 Language Features

C++11 Language Feature
Alignment support
Allowing move constructors to throw [noexcept]
Bidirectional fences
C99 preprocessor
Data-dependency ordering: atomics and memory model
Declared type of an expression
Default template arguments for function templates
Defaulted and deleted functions
Delegating constructors
Dynamic initialization and destruction with concurrency
Explicit conversion operators
Explicit virtual overrides
Expression SFINAE
Extended friend declarations
Extending sizeof
Extern templates
Forward declarations for enums
Generalized attributes
Generalized constant expressions
Inheriting constructors
Initialization of class objects by rvalues
Initializer lists
Inline namespaces
Lambdas: New wording for C++0x lambdas
Local and unnamed types as template arguments
Memory model
Minimal support for garbage collection
New character types
Non-static data member initializers
Null pointer constant
R-value references
Range-based for
Raw string literals
Right angle brackets
Sequence points
Solving the SFINAE problem for expressions

C++11 Language Feature
Standard layout types
Static assertions
Strong compare and exchange
Strongly-typed enums
Template aliases
Thread-local storage
Unicode strings literals
Universal character name literals
Unrestricted unions
User-defined literals
Variadic templates
__func__ predefined identifier
auto-typed variables
long long

C++14 Support on Linux* OS

Although C++14 is considered to be the baseline, the availability of certain C++14 features depends on the GCC* compiler version and/or the compiler option. The table below summarizes the C++14 features that are available when you use the GCC compiler option std=c++14 and link the enclave with `sgx_tcxx` as the standard C++ library.

Table 40 Supported C++14 Language Features

C++14 Language Feature
heterogeneous lookup
function return type deduction
variable template
binary literals
digit separators
generic lambdas
lambda capture expressions
attribute [[deprecated]]
aggregate member initialization
relaxed constexpr restrictions
alternate type deduction on declaration

Table 41 Supported C++14 Library Features

C++14 Library Feature
std::make_unique
std::integral_constant
std::integer_sequence
std::cbegin
std::cend
std::crbegin
std::crend
std::exchange
std::is_final
std:quoted
std::equal new overload
std::mismatch new overload
std::is_permutation new overload
standard user-defined literals

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(), wcscat_s(), strncat_s(), wcsncat_s(), strcpy_s(), wcscpy_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()