Intel® Software Guard Extensions (Intel® SGX) Enclave Common Loader API Reference

Rev 1.3
June 2021

Intel® Software Guard Extensions (Intel® SGX) Enclave Common Loader API Reference

© Intel Corporation

Table of Contents

1   Introduction

This document provides a proposal for a universal interface for loading the Intel® Software Guard Extensions (Intel® SGX) enclaves on different operating systems. It proposes to put an abstraction layer between a loader function, which creates an image of an enclave, and a lower level APIs, which create an interface to kernel modules to load the enclave contents into the Enclave Page Cache (EPC) memory (including the creation of the enclave SECS and the initialization of the enclave). The lower level APIs are currently implemented differently on various operating systems and environments.

1.1   Terminology

SGX	Software Guard Extensions
EDMM	Enclave Dynamic Memory Management
EPC	Enclave Page Cache – memory reserved for SGX enclaves
EINITTOKEN	EINIT Token – an Enclave Launch Token
SECS	Secure Enclave Control Structure
SIGSTRUCT	Enclave Signature Structure

Table 1-1: Terminology

2   Overview

This document covers the following topics:

- Enclave Loading APIs – APIs used in creating, loading, initializing, and deleting SGX enclaves
- Enclave Management APIs – APIs used in configurating enclaves or requesting enclave information
- Data Structures – data types and structures used in the APIs.

3 Enclave Loading APIs

3.1 Enclave Creation

The **enclave_create** API provides a generic API to create an enclave.

Syntax
```
void* cdecl enclave_create(
  _In_opt_  void*       base_address,
  _In_      size_t      virtual_size,
  _In_      size_t      initial_commit,
  _In_      uint32_t    type,
  _In_      const void* info,
  _In_      size_t      info_size,
  _Out_opt_ uint32_t*   enclave_error
);
```

Parameters

base_address [in, optional]

> Optional preferred base address for the enclave. Specify *NULL* to have the base address assigned by the interface.

virtual_size [in]

> Virtual address range of the enclave in bytes. This is the amount of virtual memory to reserve for the enclave to use.

initial_commit [in]

> Amount of physical memory to reserve for the initial load of the enclave in bytes. If the system does not have enough physical memory to commit to the enclave load, the function fails.

> The interface may commit larger size if required to adhere to architectural restrictions and may reserve physical pages. Any memory that remains unused after you initialize the enclave by calling **enclave_initialize** is returned as a list of empty pages.

type [in]

> Architecture type of the enclave that you want to create.

> The value must be of type ENCLAVE_TYPE (see 4.2 ENCLAVE_TYPE Enumeration).

info [in]

> Pointer to the architecture-specific information to use for the enclave creation.

> For **ENCLAVE_TYPE_SGX1** and **ENCLAVE_TYPE_SGX2**, you must specify a pointer to the **enclave_create_sgx_t** structure (see section 5.5).

info_size [in]

Length of the structure that the *info* parameter points to, in bytes. The length must match the structure that is relevant for the enclave architecture, otherwise it is set to 0.

enclave_error [out, optional]

Optional pointer to a variable that receives an enclave error code. Possible values are described in the ENCLAVE_ERROR Enumeration section.

Return value

If the function succeeds, the return value is the base address of the created enclave.

If the function fails, the return value is *NULL*. The extended error information is stored in the *enclave_error* parameter if used.

Remarks:

This function must be called to start loading the enclave. You should retain the return value to use it in other Enclave Common Loader APIs.

The **enclave_create_ex** API provides a generic API for creating an enclave with extended features.

Syntax
```
void* cdecl enclave_create_ex(
    _In_opt_  void*          base_address,
    _In_      size_t         virtual_size,
    _In_      size_t         initial_commit,
    _In_      uint32_t       type,
    _In_      const void*    info,
    _In_      size_t         info_size,
    _In_      const uint32_t ex_features,
    _In_      const void*    ex_features_p[32],
    _Out_opt_ uint32_t*      enclave_error
);
```

Parameters

base_address [in, optional]

Optional preferred base address for the enclave. Set to *NULL* to have the base address assigned by the interface.

virtual_size [in]

Virtual address range of the enclave in bytes. This is the amount of virtual memory to reserve for the enclave usage.

initial_commit [in]

Amount of physical memory to reserve for the initial loading of the enclave in bytes. If the system does not have enough physical memory to commit to the enclave loading, the function fails.

The interface may commit larger size if required to adhere to architectural restrictions, and it may reserve physical pages. Any memory that remains unused after you initialize the enclave by calling **enclave_initialize** is returned as a list of empty pages.

type [in]

> Architecture type of the enclave that you want to create.
>
> The value must be of type ENCLAVE_TYPE (see 4.2 ENCLAVE_TYPE Enumeration).

info [in]

> Pointer to the architecture-specific information that is used for the enclave creation.
>
> For **ENCLAVE_TYPE_SGX1** and **ENCLAVE_TYPE_SGX2**, you must specify a pointer to the **enclave_create_sgx_t** structure (see section 5.5).

info_size [in]

> Length of the structure that the *info* parameter points to, in bytes. The length must match the structure that is relevant for the enclave architecture, otherwise it is set to 0.

ex_features *[in]*

> Bitmask that defines the extended features to activate on the enclave creation.
>
> Bit [0] – enable enclave elrange setting.
>
> Bits [1:31] – 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 the enclave_elrange_t structure(see section [enclave_elrange_t Structure](#)).
>
> ex_features_p[1:31] – reserved, must be NULL.

enclave_error [out, optional]

> Optional pointer to a variable that receives an enclave error code. Possible values are described in the ENCLAVE_ERROR Enumeration section.

Return value

> If the function succeeds, the return value is the base address of the created enclave.
>
> If the function fails, the return value is *NULL*. The extended error information is stored in the *enclave_error* parameter if used.

Remarks:

> This function must be called to start loading the enclave. You should retain the return value to use it in other Enclave Common Loader APIs.

3.2　Loading Enclave Data

The **enclave_load_data** API provides a generic API to add pages to an enclave created by the **enclave_create** API.

Syntax
```
size_t cdecl enclave_load_data(
  _In_       void*        target_address,
  _In_       size_t       target_size,
  _In_opt_   const void*  source_buffer,
  _In_       uint32_t     data_properties,
  _Out_opt_  uint32_t*    enclave_error
);
```

Parameters

target_address [in]

Address in the enclave where to load the data.

target_size [in]

Size of the range to load in the enclave, in bytes. This value must be whole-number multiple of the page size.

source_buffer [in, optional]

Optional pointer to the data to load into the enclave.

If used, the size of the buffer must be identical to the *target_size*. If *NULL*, the loaded data is all zeros.

data_properties [in]

Properties of the pages to add to the enclave, including access permissions and others. For example, specific properties for the Intel® SGX include the page type and whether the loading data should be measured.

The value must be bitwise OR of the ENCLAVE_PAGE_PROPERTIES enumeration (see section 5.3)

enclave_error [out, optional]

Optional pointer to a variable that receives an enclave error code. Possible values are described in the **ENCLAVE_ERROR** enumeration (see section 5.1).

Return value

Return value is the number of bytes that was loaded into the enclave.

If the number differs from the *target_size* parameter value, this indicates an error . The extended error information is stored in the *enclave_error* parameter if used.

Remarks

The *target_address* must be specified within a previously created enclave memory range.

3.3   Enclave Initialization

The **enclave_initialize** API provides a generic API to initialize a created enclave with loaded data.

Syntax

```
bool cdecl enclave_initialize(
  _In_      void*       base_address,
  _In_      const void*  info,
  _In_      size_t      info_size,
  _Out_opt_ uint32_t*    enclave_error
);
```

Parameters

base_address [in]

> Enclave base address as returned from the **enclave_create** API.

info [in]

> Pointer to the architecture-specific information to use for the enclave initialization.
>
> For **ENCLAVE_TYPE_SGX1** and **ENCLAVE_TYPE_SGX2**, you must specify a pointer to the **enclave_init_sgx_t** structure (see section 5.6).

info_size [in]

> Length of the structure that the *info* parameter points to, in bytes. The length must match the structure that is relevant for the enclave architecture, otherwise it is set to 0.

enclave_error [out, optional]

> Optional pointer to a variable that receives an enclave error code. Possible values are described in the **ENCLAVE_ERROR** Enumeration section.

Return value

> If the function succeeds, the return value is nonzero.
>
> If the function fails, the return value is 0 and the extended error information is stored in the *enclave_error* parameter if used.

3.4   Enclave Deletion

The **enclave_delete** API provides a generic API to delete an existing enclave.

Syntax

```
bool cdecl enclave_delete(
  _In_      void*       base_address,
  _Out_opt_ uint32_t*    enclave_error
);
```

Parameters

base_address [in]

> Enclave base address as returned from the **enclave_create** API.

enclave_error [out, optional]

> Optional pointer to a variable that receives an enclave error code. Possible values are described in the **ENCLAVE_ERROR** Enumeration section.

Return value

Intel® Software Guard Extensions (Intel® SGX) Enclave Common Loader API Reference

If the function succeeds, the return value is nonzero.

If the function fails, the return value is 0 and the extended error information is stored in the *enclave_error* parameter if used.

4   Enclave Management APIs

4.1   Enclave Get Information

The **enclave_get_information** API provides an extensible API to get specific information from an enclave.

Syntax
```
bool cdecl enclave_get_information(
    _In_      void*       base_address,
    _In_      uint32_t    info_type,
    _Out_     void*       output_info,
    _In_Out_  size_t*     output_info_size,
    _Out_opt_ uint32_t*   enclave_error
);
```

Parameters

base_address [in]

> Enclave base address as returned from the enclave_create API.

info_type[in]

> Type of the requested information. Supported types are provided in the 0

> ENCLAVE_INFO_TYPE Enumeration section:
> * ENCLAVE_LAUNCH_TOKEN: provides a launch token. A valid Launch Token can be provided only for an initialized enclave.

output_info[in]

> Pointer to the information returned by the API.

output_info_size[in, out]

> Size of the output buffer, in bytes. If the function succeeds, the number of bytes is returned in the output_info. If the function fails with, ENCLAVE_INVALID_SIZE, the required size is returned.

enclave_error [out, optional]

> Optional pointer to the variable that receives an enclave error code. Possible values are described in the **ENCLAVE_ERROR** enumeration (section 5.1).

Return value

> If the function succeeds, the return value is nonzero. If the function fails, the return value is zero and the extended error information is stored in the *enclave_error* parameter.

> If the function fails with the ENCLAVE_INVALID_SIZE error, the reuested size is stored in the *output_info_size* parameter

> If the function fails with the ENCLAVE_NOT_SUPPORTED error, the operation is not supported for the defined *info_type.*

> If the function receives an unknown *info_type*, the ENCLAVE_NOT_SUPPORTED error is returned.

For specific *info_type* values:
- ENCLAVE_LAUNCH_TOKEN may fail with the following errors:
  - ENCLAVE_NOT_INITIALIZED : a launch token may only be returned for an initialized enclave
  - ENCLAVE_NOT_SUPPORTED: for platforms where the launch token is not provided to user space, the API returns ENCLAVE_NOT_SUPPORTED

Remarks

This API returns specific enclave information based on the *info_type* parameter.

4.2   Enclave Set Information

The **enclave_set_information** API provides an extensible API to set specific information for an enclave.

Syntax
```
bool cdecl enclave_set_information(
    _In_      void*       base_address,
    _In_      uint32_t    info_type,
    _In_      void*       input_info,
    _In_      size_t      input_info_size,
    _Out_opt_ uint32_t*   enclave_error
);
```

Parameters

base_address [in]

Enclave base address as returned from the enclave_create API.

info_type[in]

Type of the requested information. Supported types are provided in 0

ENCLAVE_INFO_TYPE Enumeration:
- ENCLAVE_LAUNCH_TOKEN: provides a launch token. A valid Launch Token can be provided only for a **uninitialized** enclave.
- ENCLAVE_GET_LAUNCH_TOKEN_FUNCTION: provides a function pointer to a function that is able to obtain Launch Tokens.

input_info[in]

Pointer to information provided to the API.

input_info_size[in]

Size of the information, in bytes, provided in the *input_info* parameter.

enclave_error [out, optional]

Optional pointer to a variable that receives an enclave error code. Possible values are described in the **ENCLAVE_ERROR** enumeration (section 5.1).

Return value

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero and the extended error information is stored in the *enclave_error* parameter.

Intel® Software Guard Extensions (Intel® SGX) Enclave Common Loader API Reference

The ENCLAVE_INVALID_SIZE error indicates an invalid value of the *input_info_size*.

If the function fails with the ENCLAVE_NOT_SUPPORTED error, the operation is not supported for the defined *info_type.*

If the API receives an unknown *info_type*, the ENCLAVE_NOT_SUPPORTED error is returned.

For specific *info_type* values:
- ENCLAVE_LAUNCH_TOKEN may fail with:
    - o ENCLAVE_ALREADY_INITIALIZED : a launch token may only be provided for an uninitialized enclave (the API is not useful if the enclave is initialized)
    - o ENCLAVE_NOT_SUPPORTED: for platforms where the launch token is not provide from user space, the API returns ENCLAVE_NOT_SUPPORTED
- ENCLAVE_GET_LAUNCH_TOKEN_FUNCTION may fail with:
    - o ENCLAVE_INVALID_PARAMETER : If *input_info* is NULL and *input_info_size* is not 0, or if *input_info* is not NULL and *input_info_size* is not the size of the function pointer type sgx_get_launch_token_func_t .

Remarks

This API sets specific enclave information based on the *info_type* parameter.

For *info_type* ENCLAVE_GET_LAUNCH_TOKEN_FUNCTION, this function sets a user-specified function pointer of type sgx_get_launch_token_func_t as the way to obtain launch tokens. If not NULL, when the Enclave Common Loader obtains launch tokens, it will call this user-specified function to obtain launch tokens. Enclave Common Loader users can set this function pointer to NULL (with size 0) to indicate that they would like to revert to the traditional mechanisms to obtain launch tokens.

5   Data Types

5.1   ENCLAVE_ERROR Enumeration

Enclave error enumeration describes the result of an enclave action and the specific error occurred if any.

Enumeration	Value	Meaning
ENCLAVE_ERROR_SUCCESS	0x00000000	No error
ENCLAVE_NOT_SUPPORTED	0x00000001	Enclave type not supported, Intel® SGX is not supported, the Intel® SGX device is not present, or the AESM Service is not running.
ENCLAVE_INVALID_SIG_STRUCT	0x00000002	SGX – SIGSTRUCT contains an invalid value
ENCLAVE_INVALID_SIGNATURE	0x00000003	SGX – invalid signature or the SIGSTRUCT value
ENCLAVE_INVALID_ATTRIBUTE	0x00000004	SGX – invalid SECS attribute
ENCLAVE_INVALID_MEASUREMENT	0x00000005	SGX – invalid measurement
ENCLAVE_NOT_AUTHORIZED	0x00000006	Enclave not authorized to run. For example, the enclave does not have a signing privilege required for a requested attribute.
ENCLAVE_INVALID_ENCLAVE	0x00000007	Address is not a valid enclave
ENCLAVE_LOST	0x00000008	SGX – enclave is lost (likely due to a power event)
ENCLAVE_INVALID_PARAMETER	0x00000009	Invalid Parameter (unspecified) – may occur due to a wrong length or format type
ENCLAVE_OUT_OF_MEMORY	0x0000000a	Out of memory. May be a result of allocation failure in the API or internal function calls
ENCLAVE_DEVICE_NO_RESOURCES	0x0000000b	Out of EPC memory
ENCLAVE_ALREADY_INITIALIZED	0x0000000c	Enclave has already been initialized
ENCLAVE_INVALID_ADDRESS	0x0000000d	Address is not within a valid enclave Address has already been committed
ENCLAVE_RETRY	0x0000000e	Please retry the operation – an unmasked event occurred in EINIT
ENCLAVE_INVALID_SIZE	0x0000000f	Invalid size

ENCLAVE_NOT_INITIALIZED	0x00000010	Enclave is not initialized – the operation requires an initialized enclave
ENCLAVE_SERVICE_TIMEOUT	0x00000011	The launch service timed out when attempting to obtain a launch token. Check to ensure that the AESM service is running and accessible.
ENCLAVE_SERVICE_NOT_AVAILABLE	0x00000012	The launch service is not available when attempting to obtain a launch token. Check to ensure that the AESM service is running.
ENCLAVE_MEMORY_MAP_FAILURE	0x00000013	Failed to reserve memory for the enclave.
ENCLAVE_UNEXPECTED	0x00001001	Unexpected error in the API

5.2   ENCLAVE_TYPE Enumeration

The enclave type enumeration describes the architecture type of the enclave.

Value	Meaning
ENCLAVE_TYPE_SGX1 0x00000001	Enclave for the Intel® Software Guard Extensions (Intel® SGX) architecture version 1.
ENCLAVE_TYPE_SGX2 0x00000002	Enclave for the Intel® Software Guard Extensions (Intel® SGX) architecture version 2 or SGX2. SGX2 adds Enclave Dynamic Memory Management instructions, which permit an application or the enclave to modify page attributes and add or remove pages from the enclave after EINIT.

5.3   ENCLAVE_PAGE_PROPERTIES Enumeration

Value	Meaning
ENCLAVE_PAGE_READ 0x00000001	Enables read access to the committed region of pages.
ENCLAVE_PAGE_WRITE 0x00000002	Enables write access to the committed region of pages.
ENCLAVE_PAGE_EXECUTE 0x00000004	Enables execute access to the committed region of pages.

ENCLAVE_PAGE_THREAD_CONTROL 0x00000100	Page contains a thread control structure.
ENCLAVE_PAGE_UNVALIDATED 0x00001000	Provided page contents are excluded from measurement and content validation.

5.4 ENCLAVE_INFO_TYPE Enumeration

The enclave type enumeration describes the information type.

Value	Meaning
ENCLAVE_LAUNCH_TOKEN 0x00000001	Get or set the launch token.
ENCLAVE_GET_LAUNCH_TOKEN_FUNCTION 0x00000002	Used to set the function to be used for obtaining launch tokens.

5.5 enclave_create_sgx_t Structure

Contains architecture-specific information to use for enclave creation when the enclave type is ENCLAVE_TYPE_SGX1 or ENCLAVE_TYPE_SGX2, which specifies an enclave for the Intel® Software Guard Extensions (Intel® SGX) architecture.

Syntax

```
typedef struct enclave_create_sgx_t {
  uint8_t    secs[4096];
} enclave_create_sgx_t;
```

Members

secs

   The Intel SGX enclave control structure (SECS) to use for the enclave creation.

5.6 enclave_init_sgx_t Structure

Contains architecture-specific information used to initialize an enclave when the enclave type is ENCLAVE_TYPE_SGX1 or ENCLAVE_TYPE_SGX2, which specifies an enclave for the Intel® Software Guard Extensions (Intel® SGX) architecture.

Syntax

```
typedef struct enclave_init_sgx_t {
  uint8_t    sigstruct[1808];
} enclave_init_sgx_t;
```

Members

Intel® Software Guard Extensions (Intel® SGX) Enclave Common Loader API Reference

sigstruct

>The Intel SGX Enclave Signature Structure (SIGSTRUCT) to use for the enclave initialization.

5.7   enclave_sgx_attr_t Structure

Contains architecture-specific attribute information used to obtain an EINIT TOKEN for an enclave of type ENCLAVE_TYPE_SGX1 or ENCLAVE_TYPE_SGX2, which specifies an enclave for the Intel® Software Guard Extensions (SGX) architecture.

Syntax
```
typedef struct enclave_sgx_attr_t {
  uint8_t     attributes[16];
} enclave_sgx_attr_t;
```

Members

attributes

>SGX enclave's attribute information.

5.8   enclave_sgx_token_t Structure

Contains architecture-specific information used to initialize an enclave of type ENCLAVE_TYPE_SGX1 or ENCLAVE_TYPE_SGX2, which specifies an enclave for the Intel® Software Guard Extensions (SGX) architecture.

Syntax
```
typedef struct enclave_sgx_token_t {
  uint8_t     token[304];
} enclave_sgx_token_t;
```

Members

token

>SGX Launch Token (EINITTOKEN) for the enclave initialization.

5.9   sgx_get_launch_token_func_t

Pointer to a function used to obtain a launch token of type enclave_sgx_token_t. For an enclave of type ENCLAVE_TYPE_SGX1 or ENCLAVE_TYPE_SGX2, which specifies an enclave for the Intel® Software Guard Extensions (SGX) architecture, instead of relying on the typical mechanisms used by the Enclave Common Loader to obtain a launch token, users may provide their own function to obtain launch tokens. The enclave common library will call this user-provided function instead of its traditional means to obtain launch tokens.

Syntax
```
typedef uint32_t(COMM_API* sgx_get_launch_token_func_t)
(
    _In_  const enclave_init_sgx_t* css,
    _In_  const enclave_sgx_attr_t* attr,
    _Out_ enclave_sgx_token_t* token
```

Intel® Software Guard Extensions (Intel® SGX) Enclave Common Loader API Reference

```
                                                    );
```

Parameters

css [in]

> Pointer to an enclave_init_sgx_t structure.

attr [in]

> Pointer to an enclave_sgx_attr_t structure.

token [out]

> Pointer to an enclave_sgx_token_t structure.

Return value

> If the function succeeds, it should return value is ENCLAVE_ERROR_SUCCESS.

> If the function fails, it should return a value that corresponds to a different ENCLAVE_ERROR as defined in the ENCLAVE_ERROR Enumeration.

5.10   enclave_elrange_t Structure

Contains architecture-specific information to use for the enclave creation with extended feature when the enclave type is ENCLAVE_TYPE_SGX1 or ENCLAVE_TYPE_SGX2, which specifies an enclave for the Intel® Software Guard Extensions (Intel® SGX) architecture.

Syntax
```
typedef struct enclave_elrange{
    uint64_t enclave_image_address;
    uint64_t elrange_start_address;
    uint64_t elrange_size;
}enclave_elrange_t;
```

Members

enclave_image_address

> The base address of the enclave image file.

elrange_start_address

> The base address of the enclave address range.

elrange_size

> The size of the enclave address range.

# 6	Disclaimer and Legal Information

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

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

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

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

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

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

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

Optimization Notice

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

Notice revision #20110804

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

© Intel Corporation

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

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