

# Intel® Software Guard Extensions (Intel® SGX) Protected Code Loader (PCL) for Linux*

## User Guide

**26 February 2018**
**Revision 1.0.1**



**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.

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 and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

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

# Contents



# Tables

# Figures

# 1 Introduction

## 1.1 General

The Intel® Software Guard Extensions (Intel® SGX) Protected Code Loader (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**: The enclave shared object (`.so`) is encrypted at build time. It is decrypted at enclave load time.

## 1.2 Terminology

**Table 1: Terminology**

| Term | Description |
|---|---|
| Intel® SGX | Intel® Software Guard Extensions |
| Intel® SGX PCL | Protected Code Loader |
| Enclave Application | Ring 3 application that utilizes one or more Intel® SGX enclaves |
| IP | Intellectual Property |
| IP Binary | IP code or data in the enclave binary image |
| Non IP Binary | Code or data in the enclave image which are not IP |
| ELF | Executable Linkable Format. Linux executable application. |
| Section | ELF format binary file section. |
| IP Section | Section which includes IP binary |
| Non IP Section | Section which does not include IP binary |
| ISV | Independent Software Vendor |
| so | Linux Shared Object file |
| IP Enclave | Also referred to as 'Encrypted Enclave'.<br>Main/product ISV enclave. Contains the ISV's encrypted IP. |
| Sealing enclave | Also known as 'Decryption-Key Provisioning-Enclave'.<br>Auxiliary ISV enclave. Provisions the decryption-key to the platform and seals it. |
| CentOS | CentOS 7.3.1611 64bit |
| Ubuntu | Ubuntu* Desktop -16.04-LTS 64bit |
| RHEL | Red Hat Enterprise Linux Server release 7.3 64bit |

## 1.3 References

**Table 2: References**

| # | Title | Description |
|---|---|---|
| 1 | Intel® Software Guard Extensions Remote Attestation End-to-End Example | Article describes Intel® SGX Remote Attestation flow in detail through an example end-to-end application that was developed at Intel. |
| 2 | Intel® Software Guard Extensions Developer Guide | Intel® SGX SDK developer guide provides guidance on how to develop robust application enclaves based on Intel® Software Guard Extensions technology. |
| 3 | Intel® Software Guard Extensions Developer Reference | Intel® SGX developer reference covers tutorials, tools, and API references, as well as sample code. |

## 1.4 Legal considerations

The Intel® SGX PCL add-on uses code snippets from OpenSSL1.1.0g and from the Intel® SGX PSW and Intel® SGX SDK.

## 1.5 Package Content

The Intel® SGX PCL add-on is a separate add-on to Intel® SGX SDK branches sgx_2.0, sgx_2.1 and sgx_2.1.1. It contains the components listed/summarized in Table 3.

**Table 3: Intel® SGX PCL add-on package content**

| Component Name | Description |
|---|---|
| libsgx_pcl.a / libsgx_pclsim.a | Trusted libraries to be added to the enclave at link time for HW or simulation modes, respectively. |
| sgx_encrypt | A tool that encrypts the ISV's enclave. ISVs integrate it into their build flow such that it runs between the link and sign phases. |
| sgx_pcl_guid.h | Include file used by the Sealing Enclave to create the sealed key blob and by the encryption tool to update the encrypted IP enclave. |
| SampleEnclave | A sample project showing how Intel® SGX PCL can be integrated into existing SampleEnclave. |
| Source code | Open source code of all tools and libraries. |
| **Note:** All the libraries/executables are built for Linux X64 configuration. | |

# 2 Architecture Overview

## 2.1 Introduction

This section introduces the Intel® SGX PCL for Linux. The following topics are covered:

- Build time
- Run time
- Comparison with standard flow
- Security considerations

## 2.2 Build Time

Figure 1 shows the Intel® SGX PCL build flow.

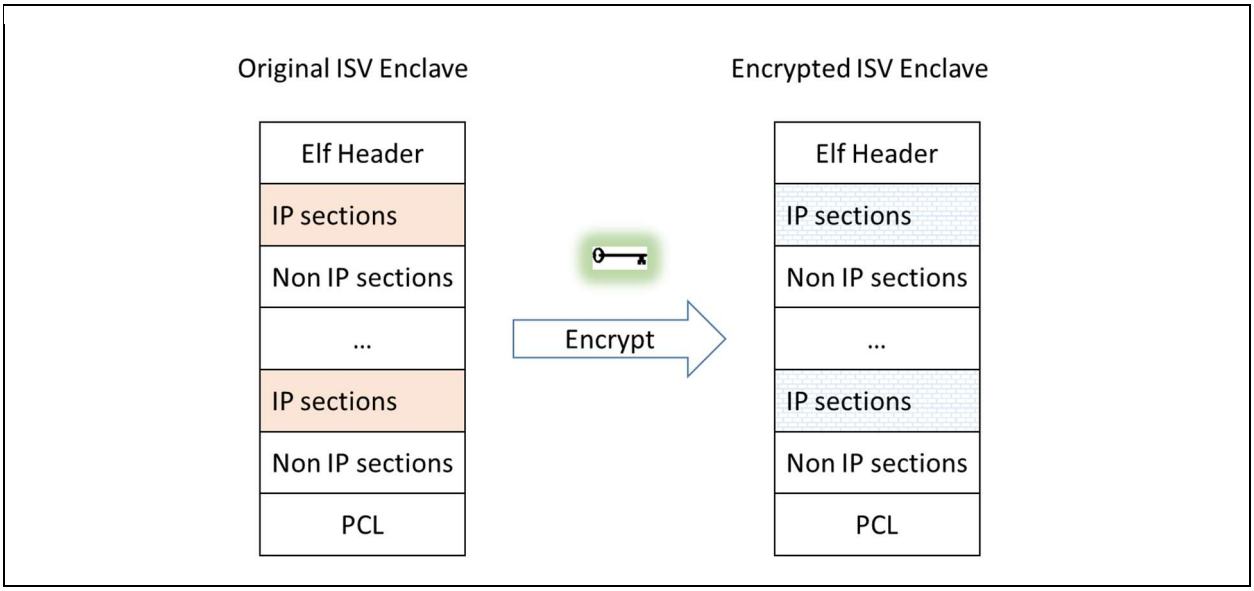


**Figure 1: Intel® SGX PCL Build Flow**

1. The Intel® SGX PCL library is linked into the ISV Intel® SGX IP Enclave.
2. Before the ISV's IP Enclave is signed, the linked shared object is modified such that ELF sections that contain 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's responsibility and is out of scope for this document.

## 2.3 Run Time

### 2.3.1 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 that user local machine, and use it as an input for the Intel® SGX PCL. For this, the ISV must devise a second enclave, the 'Sealing Enclave'. Figure 2 shows this flow.

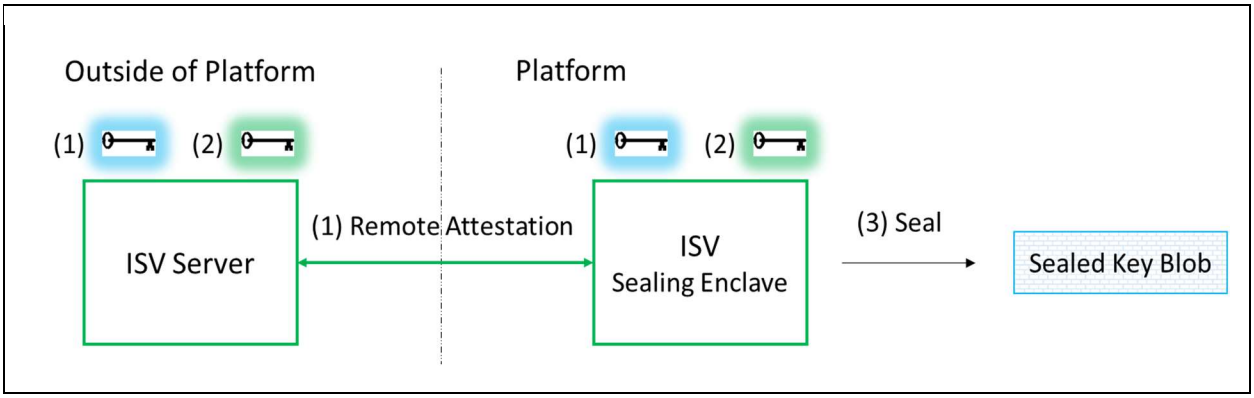


**Figure 2: ISV Sealing Enclave Flow**

The ISV's Sealing Enclave performs the following:

1. Uses **existing standard** Intel® SGX SDK **Remote Attestation** to generate a secure session with the ISV server. Details at "Remote Attestation" below. (Light blue key illustrates session keys)
2. Receives the decryption key from the ISV server in a secured way.
   Details at "Decryption Key Provisioning" below. (Green key illustrates decryption key)
3. Uses **existing standard** Intel® SGX SDK **Sealing** example to generate the sealed key and store it locally.

**Notes**:

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

### 2.3.2 ISV IP Enclave

Figure 3 shows the enclave loading flow.

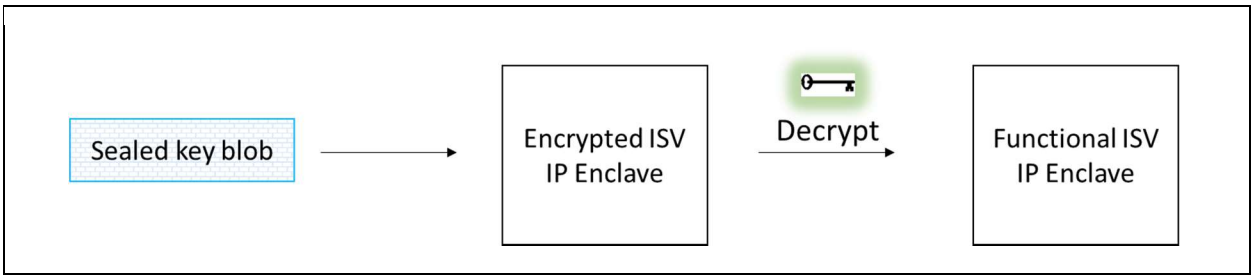


**Figure 3: ISV IP Enclave Loading Flow**

The ISV's IP Enclave performs the following:

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.

## 2.4 Comparison with Standard Flow

Table 4 summarizes the differences (in blue) between IP Enclave load flows with and without the Intel® SGX PCL:

**Table 4: Comparison of enclave load flows**

| Step | Standard Flow (no Intel® SGX PCL) | Intel® SGX PCL Flow |
|---|---|---|
| Build Time | 1. **Link**: ISV's archives and objs are linked into Enclave.so<br>2. **Sign**: Enclave.so is signed to generate Enclave.signed.so | 1. **Link**: ISV's archives, objs, and libsgx_pcl.a are linked into IPEnclave.so<br>2. Encrypt IPEnclave.so → IPEnclave.so.enc<br>3. **Sign**: IPEnclave.so.enc is signed to generate IPEnclave.signed.so |
| Enclave Load | 1. Enclave application loads the enclave using sgx_create_enclave<br>2. sgx_create_enclave performs an implicit ecall.<br>3. First ecall initiates enclave runtime initialization flow. | 1. Enclave application gets the sealed decryption key<br>2. Enclave application loads the enclave using sgx_create_encrypted_enclave, providing the sealed decryption key<br>3. sgx_create_encrypted_enclave performs an implicit ecall.<br>4. First ecall invokes Intel® SGX PCL flow<br>5. Intel® SGX PCL unseals the sealed blob to get the decryption key.<br>6. Intel® SGX PCL decrypts the decrypted IP sections and returns the enclave to its functional state.<br>7. Continue with enclave runtime initialization flow. |
| **Note:** In simulation mode, link `libsgx_pclsim.a` and not `libsgx_pcl.a` | | |

## 2.5 Security Considerations

### 2.5.1 Sections that are Not Encrypted

The ISV must verify the ELF sections in Table 5 do *not* contain the ISV's IP. The encryption tool will **NOT** encrypt these sections.

**Table 5: Sections that are Not Encrypted**

| Section name | Description |
|---|---|
| .shstrtab | Sections' names string table. Pointed by e_shstrndx |
| .note.sgxmeta | Used by the Intel® SGX SDK |
| .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 | Sections hold the content pointed by entries with index DT_SYMTAB, DT_STRTAB and DT_REL 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. |

### 2.5.2 Cryptography

#### 2.5.2.1 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.

#### 2.5.2.2 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.

# 3 Integration with Intel® SGX PCL

Integrating an ISV's enclave with Intel® SGX PCL requires the ISV to apply steps 1 – 4 on the ISV's development platform:

1. Prerequisites (including Intel® SGX PSW and Intel® SGX SDK).
2. Apply modifications to the Intel® SGX SDK and Intel® SGX PSW source files.
3. Build and install the modified Intel® SGX SDK and Intel® SGX PSW.
4. Set environment variables.

Modifications to the ISV's solution:

5. Apply modifications to the IP enclave.
6. Apply modifications to the enclave application that loads the enclave(s).
7. Create an additional enclave, the Sealing Enclave.

**Note**: Steps 5 – 7 above have already been applied to the sample code in the Intel® SGX PCL release package (see `SampleCode\SampleEnclave`). It can be built and run upon completion of steps 1 – 4 above. See Building the Intel® SGX PCL Tool and Static Library below for instructions to build the encryption tool and decryption static library. See Building and running the Intel® SGX PCL Sample Code for instructions to build and run the sample code. Comparing it with the original `SampleCode\SampleEnclave` which comes as part of the Intel® SGX SDK code samples illustrates how steps 5 – 7 should be applied.

**Disclaimer**: Code in document is pseudo code / abstract. It is not secure, is not complete, and will not compile. For complete code, see the source files at the git repository.

## 3.1 Prerequisites

HW: all build flavors, except for simulation and pcl_simulation, require the platform to be Intel® SGX enabled (CPU and BIOS).

SW:

- git
- Intel® SGX PSW, Intel® SGX SDK and Intel® SGX Driver
- OpenSSL1.1.0g

### 3.1.1 git

git is used to apply a patch file to the Intel® SGX PSW and Intel® SGX SDK

#### 3.1.1.1 Ubuntu

```
$ sudo apt-get update
$ sudo apt-get install git
```

#### 3.1.1.2 CentOS & RHEL

```
$sudo yum install git
```

### 3.1.2 Intel® SGX PSW and Intel® SGX SDK

Follow instructions at linux-sgx-sdk to build and install the Intel® SGX SDK, branch sgx_2.0, sgx_2.1 or sgx_2.1.1.

On platforms that support Intel® SGX HW, you may also follow the instructions to install Intel® SGX PSW.

**Note**: Current version of the Intel® SGX PCL supports branches sgx_2.0, sgx_2.1 and sgx_2.1.1 of Intel® SGX PSW and Intel® SGX SDK.

**Note**: ISV must verify that the Intel® SGX SDK SampleCode/SampleEnclave successfully builds and runs on the ISV's platform in both simulation mode and HW mode (if HW supports Intel® SGX) before the modifications required for Intel® SGX PCL are applied. This will decrease the number of failures wrongly associated with Intel® SGX PCL.

### 3.1.3 OpenSSL1.1.0g

The build time encryption tool does not use the default OpenSSL version. It uses a newer version (1.1.0g), which must be downloaded and built.

Download OpenSSL1.1.0g from: https://www.openssl.org/source/

Build instructions: (https://wiki.openssl.org/index.php/Compilation_and_Installation)

```
$ ./config
$ make
```

**Note**: Intel® SGX PCL does not require installing OpenSSL 1.1.0g (which could possibly result in overriding the distro's default). Intel® SGX PCL only uses the headers and generated shared objects.

## 3.2 Apply modifications to Intel® SGX PSW and Intel® SGX SDK

Apply the required modifications to Intel® SGX PSW and Intel® SGX SDK source files using the supplied git patch:

```
$ cd <linux-sgx>
```

where `<linux-sgx>` is the Linux Intel® SGX PSW and Intel® SGX SDK home directory.

```
$ git apply <path_to_pcl_dir>/Tools/sgx.psw.sdk.2.1.git.diff
```

where `<path_to_pcl_dir>` is path to Intel® SGX PCL base directory (either full or relative).

**Note**: a git patch file can only be applied to a specific branch of the Intel® SGX PSW and Intel® SGX SDK. When using Intel® SGX SDK branch sgx_2.0 use sgx.psw.sdk.2.0.git.diff. When using branches sgx_2.1 or sgx_2.1.1 use sgx.psw.sdk.2.1.git.diff.

## 3.3 Rebuild and reinstall Intel® SGX PSW and Intel® SGX SDK

Follow instructions at: linux-sgx-sdk to uninstall and clean, then build and install the Intel® SGX PSW and Intel® SGX SDK.

## 3.4 Set Environment Variables

1. Set the Linux Intel® SGX PSW and Intel® SGX SDK home directory:

   ```
   $ export SGX_SDK_SRCS=< sgx_psw_sdk_sources_home_dir >
   ```

   where `< sgx_psw_sdk_sources_home_dir >` is the base directory of the Intel® SGX PSW and Intel® SGX SDK sources (that is, where the sdk and psw are located)

2. Set the OpenSSL 1.1.0g shared object directory:

   ```
   $ export OPENSSL_ROOT=< openssl_crypto_libraries_dir >
   ```

   where `< openssl_crypto_libraries_dir >` is full path to the directory where OpenSSL 1.1.0g libcrypto.so (or libcrypto.so.1.1 etc.) is located.

3. When separately building the encryption tool, Intel® SGX PCL trusted runtime library or sample enclave, set the Intel® SGX PCL root folder.

   ```
   $ export PCL_DIR=< path_to_pcl_dir >
   ```

   where `< path_to_pcl_dir >` is the full path to the main directory of Intel® SGX PCL (folder which includes the subfolder Include, Common, Tools etc.).

## 3.5 Modifications to IP Enclave

**Note**: Steps 3.5 – 3.7 have already been applied on the sample code in the Intel® SGX PCL release package (see `SampleCode\SampleEnclave`). The sample code can be built and run upon completion of steps 3.1 – 3.4 above. See Building and Running the Intel® SGX PCL Sample Code below for instructions to build and run the sample code. Comparing the attached sample code with the original `SampleCode\SampleEnclave` which comes as part of the Intel® SGX SDK code samples illustrates how steps 3.5 – 3.7 should be applied.

**Note**: See sample source code at `SampleCode\SampleEnclave`

1. Add the following 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 HW and simulation modes, respectively.
2. Add the following stage to the build flow

   ```
   ifneq ($(SGX_IPLDR),)
   PCL_ENCRYPTION_TOOL := sgx_encrypt
   PCL_KEY := key.bin
   ifeq ($(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
   ```

   - The '-d' option is added in debug mode. It informs the encryption tool not to encrypt or zero sections that must remain plain text to enable / ease debugging.
3. Modify the build flow such that sealed enclave is generated from the encrypted enclave.
4. No modifications are required to the IP Enclave source code.

## 3.6 Modifications to Enclave Application

**Note**: See sample source code at `SampleCode\SampleEnclave\App`

Required steps:

1. Get the sealed blob:
   - If file containing the sealed blob exists (for example, from previous runs) read it.
   - Else:
     - Create the Sealing Enclave.
     - Use the Sealing Enclave to provision the decryption key onto 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_encrypted_enclave` 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:
sgx_create_encrypted_enclave(
    IP_ENCLAVE_FILE_NAME,
    debug,
    &token,
    &updated,
    ip_enclave_id,
    NULL,
    sealed_key);
```

## 3.7 Sealing Enclave

**Note**: See sample source code at `SampleCode\SampleEnclave\Seal`

### 3.7.1 Decryption key provisioning

This section describes methods for the ISV to create and use the ISV's Sealing Enclave. The ISV Sealing enclave is responsible to provision the decryption key to the user local machine and seal 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.

#### 3.7.1.1 Remote Attestation

The Intel® SGX SDK Remote Attestation sample illustrates and describes in details how to initiate a remote attestation session with an ISV server.

#### 3.7.1.2 Sending the Key from ISV Server to Local Platform

The last message of remote attestation (`msg4`) includes an optional secret payload to the client (in our case, the Sealing Enclave).

Quote from [1]:

> "Remote Attestation utilizes a modified Sigma protocol to facilitate a Diffie-Hellman Key Exchange (DHKE) between the client and the server. The shared key obtained from this exchange can be used by the Service Provider to encrypt secrets to be provisioned to the client. The client enclave would then be able to retrieve the same key and use it to decrypt the secret.
> …
> After receiving the attestation status from the IAS, the Service Provider generates msg4 to the client in response to msg3. The payload of msg4 contains the attestation status and some optional values, such as the secret, which can be encrypted using the shared key derived during the DHKE."

The ISV can use any one of the following two alternatives:

- Provision the decryption key directly as the payload of that last message.
- Use the payload of that last message to provision a primary secret. Then use the primary secret to generate a secure session (for example, using TLS) between the ISV server and the Sealing enclave. Then use the secure session to securely provision the decryption key.

### 3.7.2 Sealing the key

The Intel® SGX SDK Sealing sample illustrates how to seal a secret. As default, the Intel® SGX SDK seals the secret using MRSIGNER.

### 3.7.3 Interaction with the Enclave Application

In the pseudo code above, the ISV's Sealing Enclave provides the Enclave Application with 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 choose to use their own design.

# 4 Distribution to Customers' Platforms

At runtime, the ISV's application must use the slightly modified `libsgx_urts.so`. (Generated on the ISV's development platform as described at 'Apply modifications to Intel® SGX PSW and Intel® SGX SDK' and 'Rebuild and reinstall the Intel® SGX PSW and Intel® SGX SDK' above).

If ISV does not control the Intel® SGX SW stack on the customers' platforms then ISV can use the following deployment methodology:

1. Add the Intel® SGX PCL `libsgx_urts.so` to the release package.
2. Add the path to the Intel® SGX PCL `libsgx_urts.so` to `LD_LIBRARY_PATH`

   ```
   $ export LD_LIBRARY_PATH=<path_to_urts_so_file>:$LD_LIBRARY_PATH
   ```

   where `<path_to_urts_so_file>` is the full path to the Intel® SGX PCL version of `libsgx_urts.so` file.

The Intel® SGX PSW installer inserts the original `libsgx_urts.so` (without the modifications required for Intel® SGX PCL) into `/usr/lib/`. At application load time, the dynamic linker searches for `libsgx_urts.so` in the path defined by `LD_LIBRARY_PATH` environment variable before it searches for it at `/usr/lib/`.

# 5 Release Notes

## 5.1 Supported compilation/linker options

As a rule of thumb, Intel® SGX PCL infrastructure supports all linkers / compilers that are supported by the Intel® SGX PSW and Intel® SGX SDK.

Both default gcc linker and ld-gold are supported.

## 5.2 Release code structure

1. `bin/x64` holds the generated executable tool `sgx_encrypt`
2. `Common/pcl_common.h` includes content used by files in both Intel® SGX PCL lib and encryption tool
3. `Include/sgx_pcl_guid.h` shall be used by the ISVs Sealing Enclave. It also used by the Intel® SGX PCL signing tool.
4. `lib64` holds the generated static libraries `libsgx_pcl.a` and `libsgx_pclsim.a`
5. `Tools/Encryptip` holds the sources of the encryption tool (`sgx_encrypt`) which performs the encryption at build time
6. `Tools/sgx.psw.sdk.2.0.git.diff` and `Tools/sgx.psw.sdk.2.1.git.diff` are the git patches files describing the modifications required to enable the solution to run properly. Patch `sgx.psw.sdk.2.0.git.diff` needs to be applied to Linux Intel® SGX PSW and Intel® SGX SDK branch sgx_2.0. Patch `sgx.psw.sdk.2.1.git.diff` needs to be applied to Linux Intel® SGX PSW and Intel® SGX SDK branches sgx_2.1 or sgx_2.1.1.
7. `SampleCode`

   `key.bin` is a binary file holding the dummy symmetric encryption/decryption key. ISVs must replace this with the ISVs' key.
   Key management is out of scope.

   `SampleEnclave` illustrates how ISV can port existing code to use the Intel® SGX PCL. ISVs are encouraged to compare these to the originals App and Enclave folders at `<sgx_sdk_dir>/SampleCode/SampleEnclave`. where `<sgx_sdk_dir>` is home directory for the Intel® SGX SDK, for example, `/opt/intel`

   i. `App` and `Enclave` demonstrate the modifications to enclave makefile and enclave application required to use Intel® SGX PCL.
   ii. `Seal` folder includes the sample Sealing Enclave.
8. Source holds sources for the static library that is linked to the enclave and performs the decryption at run time

   crypto:

   i. `pcl_cmac.c`, `pcl_gcm128.c`, `pcl_md32_common.h`, `pcl_modes_lcl.h`, `pcl_sha256.c`
   Files from openssl1.1.0g with modifications to enable running with Intel® SGX PCL.
   iii. `pcl_crypto.cpp`: cryptography code, calls the other functions
   iv. `pcl_crypto_internal.h`: content used by files in crypto folder
   v. `pcl_vpaes-x86_64.s`: output of `vpaes-x86_64.pl` after building openssl1.1.0g. Symbols are renamed to enable running in parallel to Intel® SGX SSL.

`pcl_entry.cpp`: Intel® SGX PCL entry point

`pcl_mem.cpp`: memory functionalities

`pcl_internal.h`: content used by multiple files in Intel® SGX PCL lib

unseal

i. `pcl_unseal_internal.h`: content used by files in unseal folder

ii. `pcl_sgx_get_key.cpp`, `pcl_tSeal.cpp`, `pcl_tSeal_internal.cpp`, `pcl_tSeal_util.cpp`
Files from Intel® SGX SDK with modifications to enable running with Intel® SGX PCL

iii. sim

1) `pcl_derive.cpp, pcl_t_instructions.cpp`
Files from Intel® SGX SDK with modifications to enable running with Intel® SGX PCL. Content is only applicable to simulation mode.

## 5.3 Building the Intel® SGX PCL Tool and Static Library

The following steps describe how to build the Intel® SGX PCL build time encryption tool and static library. ISVs can build the project according to the ISV's requirements.

**Note**: ISV must complete steps 3.1-3.4 above before the Intel® SGX PCL tool and static library can be built.

To build both Intel® SGX PCL encryption tool (`sgx_encrypt`) and Intel® SGX PCL statically linked library with default configuration, enter the following command:

```
$ cd <path_to_pcl_dir>
```

where `<path_to_pcl_dir>` is path to Intel® SGX PCL base directory

```
$ make
```

The tool `sgx_encrypt` is generated at `bin/x64` directory.

The static libraries `libsgx_pcl.a` and `libsgx_pclsim.a` are generated at `lib64` directory.

To build Intel® SGX PCL with debug information, enter the following command:

```
$ make DEBUG=1
```

To clean the files generated by previous `make` command, enter the following command:

```
$ make clean
```

**Note**: It is also possible to enter either the `Sources` or `Tools\Encryptip` folders and use the `make` command to separately build the Intel® SGX PCL static library or build time encryption tool, respectively.

**Note**: `make clean` must be run when switching between configurations. For example when switching between building with and without debug information.

## 5.4 Building and running the Intel® SGX PCL Sample Code

**Note**: ISV must build the Intel® SGX PCL (see above) before the Intel® SGX sample code can be built.

To compile and run the sample

```
$ cd <path_to_pcl_dir>/SampleCode/SampleEnclave
```

where `<path_to_pcl_dir>` is path to Intel® SGX PCL base directory

```
$ make
$ ./app
```

**Note**: See linux-sgx-sdk for instructions on building with debug information and / or building in simulation mode.

**Note**: ISVs are encouraged to compare the sample code to Intel® SGX SDK sample code as a demonstration of how the Intel® SGX PCL should be integrated into the ISV's project.