Enclave-to-Enclave Communication in Intel® Software Guard Extensions (Intel® SGX) Applications

Scope
This paper describes how two Intel® Software Guard Extensions (Intel® SGX) enclaves can securely communicate with each other when they are on the same platform (Local Attestation). This information applies to Intel SGX enabled applications for the Microsoft® Windows® and Linux® OS. The paper assumes a basic knowledge of Intel SGX. Information on Intel SGX can be found on the Intel SGX portal at: https://software.intel.com/en-us/sgx.

Local Attestation
There are situations where two or more enclaves on the same platform need to communicate with each other securely via untrusted memory space. For example, let's say that one developer creates a web logging application to collect log information and another developer creates an application that sends logging information to the first application. These applications can implement a secure form of Inter-process Communication (IPC) to send/receive log messages via enclaves. The developers need two basic elements:

- API information specific to the web logging IPC functions. Typically the developer of the web logger application shares their API definitions with the developer of the sending application.
- An attestation method that verifies the trustworthiness of enclaves and helps them to communicate with each other. The Intel® SGX SDK provides APIs used by applications to implement the attestation process. Communication between enclaves is performed through this attestation process.

**Note:** Intel SGX provides both a Local Attestation method that creates an authenticated assertion between two enclaves running on the same platform, and a Remote Attestation method to support communication between enclaves on different platforms. Local Attestation is the subject of this white paper. Information on Remote Attestation can be found on the Intel SGX portal at: https://software.intel.com/en-us/sgx.

Local Attestation
The Intel SGX Local Attestation method verifies that the platform on which the enclave resides is Intel SGX compliant, and whether the enclaves and their contents are trusted. This process is used when two enclaves on the same platform authenticate each other using the SGX
REPORT mechanism before exchanging information. After the two enclaves verify the counterpart is trustworthy, they can exchange information over a protected channel, which typically provides confidentiality, integrity, and replay protection. The enclaves can be from the same application or different applications but they must be present on the same platform.

Consider a process running in a web logging system, as shown in Figure 1. In this system, log information from Application 1 is shared with Application 2 that logs the information. Most of the data is passed from Application 1 to Application 2, but some data can pass in the reverse direction too. The two processes are in separate enclaves, with a secure connection between them to exchange secret log information. The modules need to authenticate with each other using the Local Attestation service provided by the Intel SGX architecture before passing secrets between them.

**Figure 1: Example of Local Attestation in real time system**

**Local Attestation using ECDH Key Exchange protocol**

When two enclaves of an application (or of different applications running in a system) need to exchange secret information, they do so through an untrusted zone. Since the exchange must be protected by malware attacks, an appropriate security mechanism is required to exchange secret information/keys between the enclaves.

In Intel SGX, two enclaves in a system use the Elliptic curve Diffie–Hellman (ECDH) key exchange protocol to exchange secret information between them. This is a variant of the Diffie–Hellman protocol using elliptic curve cryptography. ECDH is an anonymous key
agreement protocol that allows two parties, each having an elliptic curve public–private key pair, to establish a shared secret over an insecure channel. This shared secret may be directly used as a key, or to derive another key which can then be used to encrypt subsequent communications using a symmetric key cipher.

Assume there is an application with a source enclave and a destination enclave. 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. The application acts as the medium between the two enclaves for communication.

On 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.

The Local Attestation sample in the Intel SGX 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 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 a final library called sgx_tservice.lib, which is also in the SDK.

Figure 2 shows the usage of DH key exchange library. The Local Attestation flow consists of the following steps:

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

Figure 2: Diffie-Hellman key exchange and Local Attestation flow
Parameters passed during verification

An enclave-to-enclave communication session is established after the two enclaves verify with each other that they are on the same platform. An enclave can ask the hardware to generate a credential, also known as report, which includes cryptographic proof that the enclave exists on the platform. This report can be given to another enclave who can verify that the enclave report was generated on the same platform. The authentication mechanism used for intra-platform enclave attestation uses an asymmetric key system where only the enclave verifying the report structure and the enclave hardware creating the report know the key, which is embedded in the hardware platform.

An enclave report contains the following data:

- Measurement of the code and data in the enclave
- A hash of the public key in the ISV certificate presented at enclave initialization time
- User data
- Other security related state information
- A signature block over the above data, which can be verified by the same platform that produced the report

Secret message exchange and enclave-to-enclave call

After establishing the protected channel, session keys are used to encrypt the content in the message(s) being exchanged between the source and destination enclaves. 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 to the callee (destination) enclave.

Since one enclave cannot access memory of another enclave, all input and output parameters, including data indirectly referenced by a parameter must be marshaled across the two enclaves. The application can use a 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.

As shown in Figure 3, the `test_enclave_to_enclave_call()` is used to check whether the application has a connection established with the destination enclave. If the return value is SUCCESS, then the enclave sends the message and the sends the request to the destination enclave. The destination enclave receives the message, decrypts it, and, if successfully decrypted, returns a SUCCESS status to the application.
**Figure 3: Secret message passage between two enclaves**

**Authentication between two enclaves**

This subsection shows an example flow of how two enclaves on the same platform would authenticate each other. The flow is as follows:

1. Application A hosts enclave A and application B hosts enclave B. After the untrusted applications A and B have established a communication path between the two enclaves, enclave B sends its MRENCLAVE identity to enclave A.
   
   **Note:** Applications A and B can be the same application.

   There are two methods the application can use to retrieve the MRENCLAVE measurement for the enclave, either:

   - The application B retrieves the MRENCLAVE value from the enclave certificate for enclave B.
   - Enclave B supports an interface to export this value, which is retrieved by creating a report with a random MRENCLAVE target measurement.
2. Enclave A asks the hardware to produce a report structure destined for enclave B using the MRENCLAVE value it received from enclave B. Enclave A transmits its report to enclave B via the untrusted application. As part of his report request, enclave A can also pass in a data block of its choosing referred to as the user data. Inclusion of the user data in the report provides the fundamental primitive that enables a trusted channel to terminate in the enclave.

3. Once it has received the report from enclave A, enclave B asks the hardware to verify the report to affirm that enclave A is on the same platform as enclave B. Enclave B can then reciprocate by creating its own report for enclave A, by using the MRENCLAVE value from the report it just received. Enclave B transmits its report to enclave A.

4. Enclave A then verifies the report to affirm that enclave B exists on the same platform as enclave A.

**Local Attestation sample**

The SDK includes a sample Microsoft* Visual Studio* project called LocalAttestation. The sample includes a main application (App) with three enclaves (Enclave1, Enclave2, and Enclave3) that perform Local Attestation with each other. You can better understand the Local Attestation process by building and running this sample.

**Summary**

Two enclaves in the same application, or in different applications running on the same system, may need to exchange secret information through untrusted zones. Since secret information must be protected from malware attacks, there is a need for a mechanism to exchange secret information/keys between enclaves.

Intel SGX provides an attestation process to meet this need. Communication between two enclaves on the same platform uses Local Attestation. Local Attestation uses the Elliptic curve Diffie–Hellman (ECDH) key exchange library to establish a protected channel between two enclaves. And the hardware verifies whether they are on the same platform or not using the information sent by the enclaves.

By following the local attestation processes and using the relevant Intel SGX APIs for your situation, you can develop applications that share secrets/keys securely between enclaves on the same platform.

**References**

Intel® Software Guard Extensions (Intel® SGX)


INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL® ASSUMES NO LIABILITY WHATSOEVER AND INTEL® DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL® AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL® OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL® PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

No computer system can provide absolute security under all conditions. Built-in security features available on select Intel® processors may require additional software, hardware, services and/or an Internet connection. Results may vary depending upon configuration. Consult your system manufacturer for more details.

Intel®, the Intel® Logo, Intel® Inside, Intel® Core™, Intel® Atom™, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and/or other countries. Other names and brands may be claimed as the property of others.

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

Copyright © 2017 Intel® Corporation