System Security
Writing Intel SGX Enclave Application in Simulation Mode

Graded

Distribution: 19.11.2020
Submission due: 03.12.2020 (no extension)

1 Virtual Machine setup

We have set the necessary environment to write, execute, and test Intel SGX enclave applications in the simulation mode (`SIM`). Let's first familiarize ourselves with the VM setup.

Figure 1: Desktop environment after log-in

When you log in to the VM, you will see three directories on the desktop as shown in Figure 1:

1. linux-sgx: contains the compiled code of the SGX software development kit (SDK) and platform software (PSW). You do not need to modify this.
2. Linux-sgx-driver: contains the compiled SGX driver. Again, you do not need to modify this.
3. sgxsdk: contains all the libraries and sample code that you need to write enclave application.

Inside sgxsdk, you will find a directory called SampleCode (see Figure 2) that contains multiple projects to give you plenty of clues on how to write an SGX enclave application.

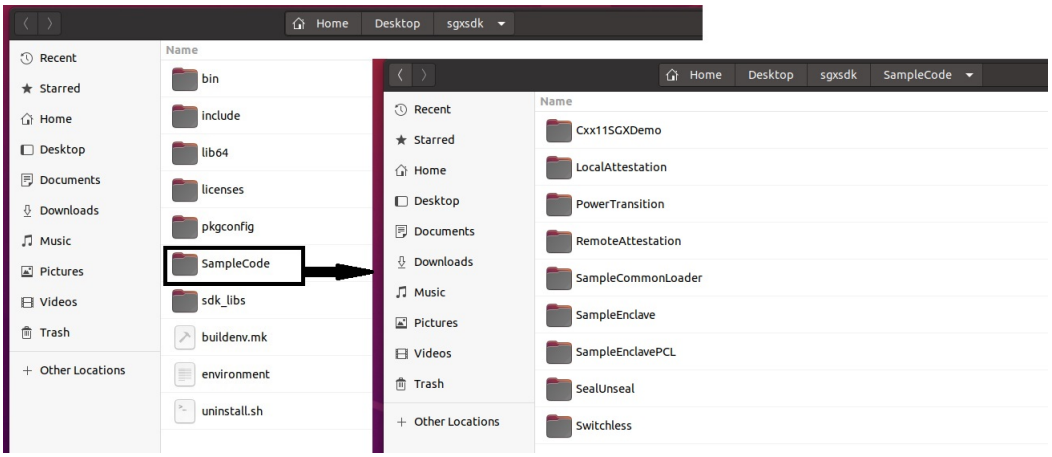

Figure 2: Inside sgxsdk directory

For example, let's look into the sample project SealUnseal that shows sealing and unsealing data between two enclaves. To compile the project, execute `make SGX_MODE=SIM` inside the SealUnseal directory that will create the executable `app`.

To execute the compiled enclave application, run: `./app`.

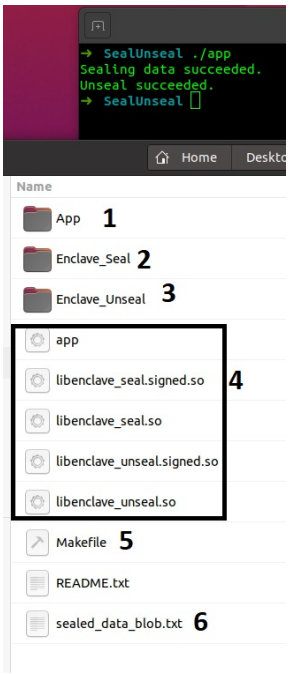

Figure 3: Structure of a sample project(SealUnseal)

Now, let's look into the structure of this project, as depicted in Figure 3.

1. The App folder contains the source code for the application that creates the enclave. Note that the App is an untrusted entity. All the IO operations are done in the App.

2. Enclave_seal is the enclave that seals data and hands it over to the App. This also

contains an enclave definition language (EDL) file that contains the OCALL and ECALL specification (more on this in Section 1.1).

3. Enclave_Unseal is the enclave that gets the sealed data from the App (via filesystem operations) and unseals the data.

4. Compiled binary and libraries. The "app" is the executable for App.

5. Makefile contains the compilation instructions.

6. The sealed data.

The Intel SGX enclave development guide is available in [1] (also included in Moodle `sgx_dev_guide.pdf`), which gives a step-by-step guide on how to create an enclave from scratch. Note that most of the part of the development guide is not necessary for this exercise as we provide a template (see Section 2.1). However, the enclave development basic is located on page 25 of [1] that is necessary for this exercise.

1.1 Important Function references from the user guide [1]

1. *Untrustd functions*, such as enclave creation and destruction, are located on page 79.

2. *Trusted function* these are the ones that you are required to use inside the enclave, located on page 80.

3. *Seal/unseal* located at page 84.

4. Crypto functions are located at page 91.

5. EDL file or the enclave definition language. The structure is described from page 36 onwards. Important point: you need to add external libraries in the edl file, e.g. `include "sgx_tcrypto.h"`. Pointer handling can be found on page 42.

2 Write your Intel SGX enclave

Take a look into other projects to know more about how to use AES, DH key exchange, etc. For this exercise, write two enclaves (E_1 and E_2) along with two untrusted apps A_1 and A_2 where E_1 and E_2 are managed by A_1 and A_2 respectively.

Figure 4 provides a two-party protocol between these two pairs of enclaves and applications. Implement this protocol. The communication between A_1 and A_2 could be implemented using a filesystem or shared memory. The apps A_1 and A_2 act as the untrusted transport between the two enclaves E_1 and E_2.

At the end of the protocol, both of the enclaves (E_1 and E_2) seals the counter i (in a file), which is at that point is 100. Make sure that you unseal the sealed data to verify that you actually sealed 100.

Tip 1: The key exchange (`sgx_ecc256_compute_shared_dhkey`) produces a 256 bit shared secret. However, the AES-CTR in SGX (`sgx_aes_ctr_encrypt` and `sgx_aes_ctr_decrypt`) only supports 128bit keys. Use the first 128 bits from the 256 bit shared key.

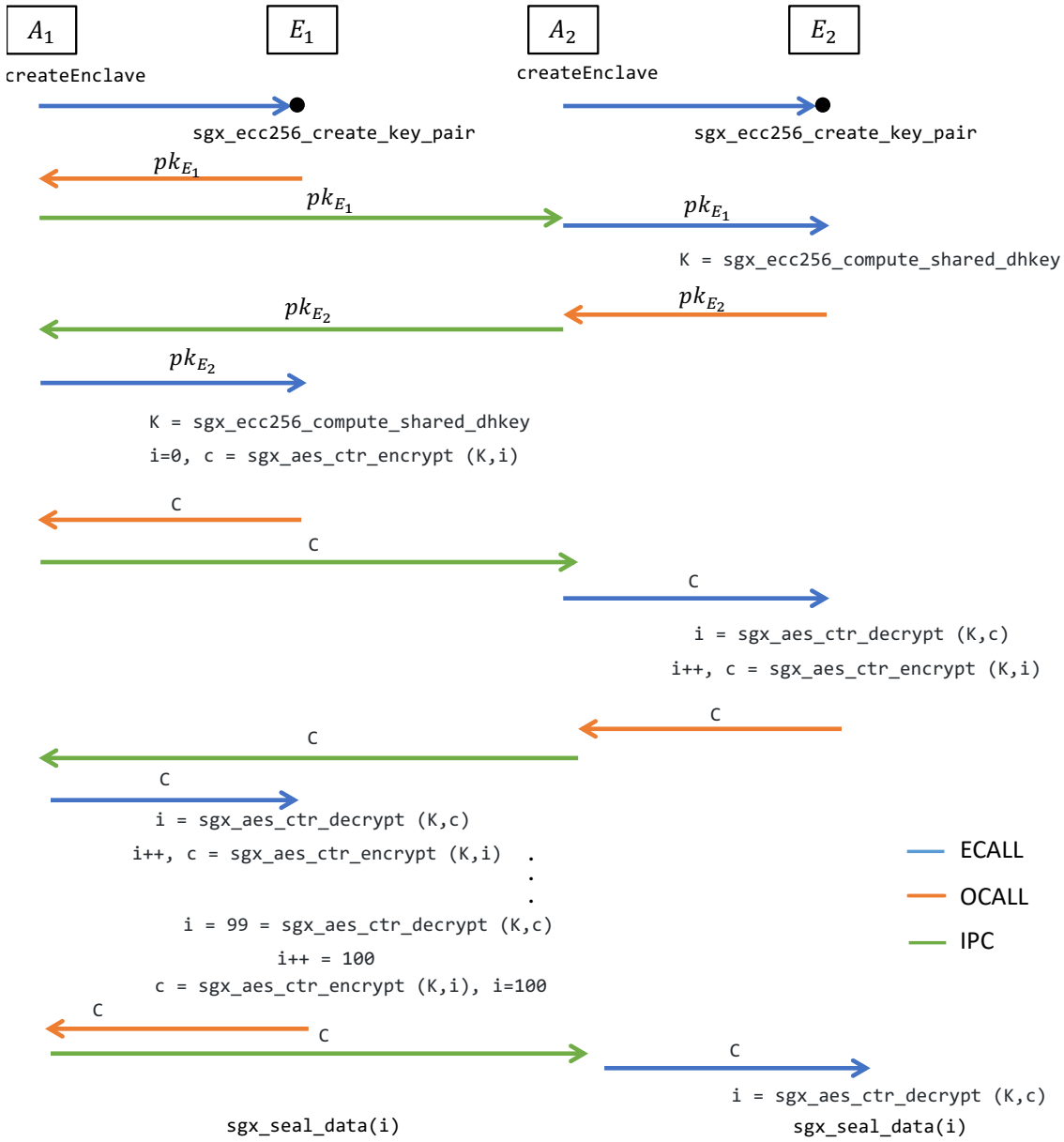

Figure 4: Protocol

Tip 2: Any changes just in the EDL files are not picked up by the incremental compilation. In such a case, do a `make clean` before `make SGX_MODE=SIM`. Make sure to include the external libraries in case you use some as the data type in the OCALL or ECALL arguments.

Tip 3: `sgx_aes_ctr_encrypt` requires a initialization vector (`*p_ctr`) and number of bits the counter to be incremented (`ctr_inc_bits`). You can initialize the IV as a zero byte array at both sides (E_1 and E_2). However, if you want a more secure communication, you should initialize the IV to a random byte array (`sgx_read_rand`) and send it across with every ciphertext. However, our grading scheme does not consider if you use this or not.

Tip 4: `sgx_ecc256_compute_shared_dhkey` produces the DH shared secret that is passed as one of the arguments to the function. The shared secret data type is `sgx_ec256_dh_shared_t` is a structure unlike the other data types used in the SGX crypto library (typically byte arrays). Access the shared secret by accessing the member `s` of the struct. For more information, look at [2] to understand the data types of SGX crypto API.

2.1 Project template

We have prepared a template for you to make the enclave programming a bit easy. You do not need to modify the make files unless you are using some external libraries. Note that the exercise could easily be done without using any external libraries. We have provided a template consists of two identical enclaves (E_1 and E_2) along with their untrusted applications (A_1 and A_2). These are placed in directories `Enclave_1` and `Enclave_2`.

2.2 How to annotate the code

Please comment on the code at the following code points:

1. The enclave applications (A_1 and A_2) read and write the shared files (both the public key and the encrypted counter).
2. The points where the enclaves (E_1 and E_2) generate their key pairs and calculate the shared secret.
3. The points where the enclaves (E_1 and E_2) encrypt and decrypt the counter.
4. The point where the enclaves (E_1 and E_2) seal the counters and the apps (A_1 and A_2) write them in a file.

2.3 Submission process

Please submit the zipped directory of the project in moodle. In case you use external libraries (which is not necessary to successfully complete this project), make sure that your `Makefile` is self-sufficient to compile your project. We cannot grade you if the compilation fails. Add a short description in the included `README.txt` file that provides clear instructions on how to run your code.

References

[1] Intel. *Intel Software Guard Extensions SDK for Linux OS*. https://01.org/sites/default/files/documentation/intel_sgx_sdk_developer_reference_for_linux_os_pdf.pdf.

[2] Intel. *linux-sgx/sgx_tcrypto.h at master . intel/linux-sgx . GitHub*. https://github.com/intel/linux-sgx/blob/master/common/inc/sgx_tcrypto.h.