Shanghai Jiao Tong University

Intel SGX

Ng Tze Kean
Student number: 721370290002

Introduction

Intel Software Guard Extensions (Intel SGX) is an Intel technology for application developers who are seeking to protect select code and data from disclosure or modification. The SGX module protects code and data used in the processor and memory by creating a trusted execution environment (TEE).

Implementation

SGX terminologies

We refer to the intel website to define the terminologies that are used in the SGX module of the intel chipset.

1. Untrusted: refers to code or construct that runs in the application environment outside the enclave.
2. Trusted: refers to code or construct that runs in the Trusted Execution Environment inside the enclave.
3. ECALL: A call from the application into an interface function within the enclave.
4. OCALL: A call made from within the enclave to the application.
5. Untrusted Run-Time System (uRTS): Code that executes outside the enclave environment and performs functions such as: Loading and manipulating an enclave (Ex: destroying an enclave). Making calls (ECALLs) to an enclave and receiving calls (OCALLs) from an enclave.
6. Trusted Run-Time System (tRTS): Code that executes within the enclave environment and performs functions such as: Receiving calls (ECALLs) from the application and making calls outside (OCALLs) the enclave. Managing the enclave itself.

Enclave

We will implement RC4 encryption in this experiment in an SGX Enclave module. There will be 3 functions to be implemented in this experiment, `ecall_rc4_sbox_init();` `ecall_rc4_generate_keystream (size_t length);` `ecall_rc4_decrypt([in, size=len] char* ciphertext, size_t len).` The key will be a `global_key` that can be accessed in the Enclave module. The ciphertext given in the experiment is passed from App to the Enclave module for decryption. The result will then be output via OCALL.

RC4

Rivest Cipher 4 (RC4) is a Stream Cipher that operates on a stream of data byte by byte. The detail of the algorithm will be briefly discussed below. References are made from here.
The encryption procedure is as follows

1. The user inputs a plain text file and a secret key.
2. The encryption engine then generates the keystream by using Key Stream Alogirhmn (KSA) and PseudoRandom Generation Algorithm (PGSA) Algorithm.
3. This keystream is now XOR with the plain text, this XORing is done byte by byte to produce the encrypted text.
4. The encrypted text is then sent to the intended receiver, the intended receiver will then decrypted the text and after decryption, the receiver will get the original plain text.

The decryption protocol is done through XOR operation on the ciphertext.

Experiment

To start with the experiment we create the EDL file to define the interface support. This is mainly for the ECALL and the OCALL calls which will be used in the Enclave. There are two parts to an EDL file. The trusted section defines the ECALLS whereas the untrusted section defines the OCALLS. While an ECALL defines entry point into the enclave, the OCALL defines the transfer of control from inside the enclave to the application to perform system calls and other I/O operations.

```
enclave {

    trusted {
        public void ecall_set_key([in, string] const char* key, size_t len);
        public void ecall_rc4_sbox_init();
        public void ecall_rc4_generate_keystream(size_t length);
        public void ecall_rc4_decrypt([in, size=len] char* ciphertext, size_t len);
    };

    untrusted {
        void ocall_print_result([in, string] char* buf, size_t len);
    };
};
```

With the defined interface, we define the Application that will interface with the Enclave. We make use of the config file for the Enclave found in some of the samples in the sandbox. We also make use of the makefile on the SDK build recommended in the instructions. Overall, the App is to first init the Enclave. We then set the encryption key defined in the instruction to be "gosecgosec". Next we make a ECALL to set the key. Once that is done, we create the S-Box and generate the keystream using the ECALL defined code. Next, we get the ciphertext and decrypt it through `ecall_rc4_decrypt`.
Within each of these functions, we can see that we define the ECALL function in the `Enclave.cpp` and also call the OCALL that is defined in `App.cpp`. This is because we want to define the protected code in the Enclave and the untrusted code in the main application which we can call.

Results

The codebase is cloned on the sandbox. To obtain the executable, we will call the make file with the following `make SGX_MODE=sim`. This will create a `app` file which can be called. Running the code, we will be prompted with the ciphertext which we will key in based on the instructions. The following result can be seen.

```
./app
>>>Generated Keystream: Result: 7A 17 32 06 15 C8 A0 FE 20 8B F0 6A 5E E4 CB FF E1
    57 25
>>>Enter the ciphertext (hex format): 1c7b53616e81ce8a45e7af3919bc94aba41258
>>>Ciphertext: 1C 7B 53 61 6E 81 CE 8A 45 E7 AF 39 19 BC 94 AB A4 12 58
>>>Result: 66 60 61 67 7B 49 6E 74 65 6C 5F 53 47 58 5F 54 45 45 7D
>>>Enter a character before exit ...
```

Since the output is a hexadecimal output, we can convert the output with a simple hexadecimal to ASCII online converter. We obtain the follow `flag{Intel_SGX_TEE}`

References

https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html
https://github.com/intel/linux-sgx

Code base

App.h

```
#ifndef _APP_H_
#define _APP_H_

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
```

```
8
9  #include "sgx_error.h"       /* sgx_status_t */
10 #include "sgx_eid.h"     /* sgx_enclave_id_t */
11
12 #ifndef TRUE
13 # define TRUE 1
14 #endif
15
16 #ifndef FALSE
17 # define FALSE 0
18 #endif
19
20 # define ENCLAVE_FILENAME "enclave.signed.so"
21
22 extern sgx_enclave_id_t global_eid;    /* global enclave id */
23
24 #if defined(__cplusplus)
25 extern "C" {
26 #endif
27
28 void edger8r_array_attributes(void);
29 void edger8r_type_attributes(void);
30 void edger8r_pointer_attributes(void);
31 void edger8r_function_attributes(void);
32
33 void ecall_libc_functions(void);
34 void ecall_libcxx_functions(void);
35 void ecall_thread_functions(void);
36
37 #if defined(__cplusplus)
38 }
39 #endif
40
41 #endif
```

App.cpp

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <assert.h>
4  #include <unistd.h>
5  #include "sgx_urts.h"
6  #include "App.h"
7  #include "Enclave_u.h"
8
9  #define ENCLAVE_FILENAME "enclave.signed.so"
10
11 sgx_enclave_id_t global_eid = 0;
12
13 void ocall_print_result(char *buf, size_t len)
14 {
15       printf("Result: ");
16       for (size_t i = 0; i < len; ++i)
17             printf("%02X ", (unsigned char)buf[i]);
18       printf("\n");
19 }
20
21 int initialize_enclave(void)
22 {
23       sgx_status_t ret = SGX_ERROR_UNEXPECTED;
24       ret = sgx_create_enclave(ENCLAVE_FILENAME, SGX_DEBUG_FLAG, NULL, NULL, &
     global_eid, NULL);
25       if (ret != SGX_SUCCESS)
26       {
27             printf("Failed to create enclave, ret code: %d\n", ret);
28             return -1;
29       }
30       return 0;
31 }
32
33 void hex_to_bytes(const char *hex, unsigned char *bytes, size_t len)
34 {
35       for (size_t i = 0; i < len; i++)
36       {
37             sscanf(&hex[2 * i], "%2hhx", &bytes[i]);
```

```
        }
}

int main(int argc, char *argv[])
{
        (void)(argc);
        (void)(argv);

        if (initialize_enclave() < 0)
        {
                printf("Enter a character before exit ...\n");
                getchar();
                return -1;
        }

        // Set the encryption key
        char key[] = "gosecgosec";
        ecall_set_key(global_eid, key, sizeof(key) - 1); // Exclude the null terminator

        // Initialize the S-box
        ecall_rc4_sbox_init(global_eid);

        // Generate and print the keystream
        printf("Generated Keystream: ");
        ecall_rc4_generate_keystream(global_eid, 19);

        // Prompt user for the ciphertext
        char hex_ciphertext[512];
        printf("Enter the ciphertext (hex format): ");
        fgets(hex_ciphertext, sizeof(hex_ciphertext), stdin);

        // Remove the newline character if present
        size_t len = strlen(hex_ciphertext);
        if (hex_ciphertext[len - 1] == '\n')
        {
                hex_ciphertext[len - 1] = '\0';
                len--;
        }

        // Convert hex string to byte array
        size_t ciphertext_len = len / 2;
        unsigned char *byte_ciphertext = (unsigned char *)malloc(ciphertext_len);
        hex_to_bytes(hex_ciphertext, byte_ciphertext, ciphertext_len);

        // Print the input ciphertext in bytes
        printf("Ciphertext: ");
        for (size_t i = 0; i < ciphertext_len; ++i)
                printf("%02X ", byte_ciphertext[i]);
        printf("\n");

        // Decrypt the ciphertext
        ecall_rc4_decrypt(global_eid, (char *)byte_ciphertext, ciphertext_len);

        // Clean up
        free(byte_ciphertext);

        // Destroy the enclave
        sgx_destroy_enclave(global_eid);
        printf("Info: SampleEnclave successfully returned.\n");
        printf("Enter a character before exit ...\n");
        getchar();
        return 0;
}
```

Enclave.h

```
#ifndef ENCLAVE_H
#define ENCLAVE_H

#include <stddef.h>  // Include size_t

#ifdef __cplusplus
extern "C" {
#endif
```

```
9
10 void ecall_set_key(const char *key, size_t len);
11 void ecall_rc4_encrypt(char *plaintext, size_t len);
12 void ecall_rc4_decrypt(char *ciphertext, size_t len);
13
14 #ifdef __cplusplus
15 }
16 #endif
17
18 #endif
```

Enclave.cpp

```
1  #include "Enclave.h"
2  #include "Enclave_t.h"
3  #include <string.h>
4  #include <stdlib.h>
5
6
7  /*We obtain the init and the crypt from public repositories*/
8
9  // RC4 Generate SBox
10 void rc4_init(unsigned char *s, const unsigned char *key, size_t key_len)
11 {
12       int i, j = 0;
13       unsigned char k[256];
14       unsigned char tmp;
15
16       for (i = 0; i < 256; i++)
17       {
18             s[i] = (unsigned char)i;
19             k[i] = key[i % key_len];
20       }
21
22       for (i = 0; i < 256; i++)
23       {
24             j = (j + s[i] + k[i]) % 256;
25             tmp = s[i];
26             s[i] = s[j];
27             s[j] = tmp;
28       }
29 }
30
31 // RC4 Generate Keystream and apply XOR for encryption/decryption
32 void rc4_crypt(unsigned char *s, unsigned char *data, size_t len, unsigned char *
       keystream)
33 {
34       int i = 0, j = 0;
35       size_t k;
36       unsigned char tmp;
37
38       for (k = 0; k < len; k++)
39       {
40             i = (i + 1) % 256;
41             j = (j + s[i]) % 256;
42             tmp = s[i];
43             s[i] = s[j];
44             s[j] = tmp;
45             unsigned char ks_byte = s[(s[i] + s[j]) % 256];
46             if (keystream != NULL)
47             {
48             keystream[k] = ks_byte;
49             }
50             data[k] ^= ks_byte;
51       }
52 }
53
54 /*We create the global variable such that it can be used within the module*/
55
56 // Global variables for RC4
57 unsigned char global_key[256];
58 size_t global_key_len;
59 unsigned char s[256]; // S-box
60
61 /*We define the implementation of the interface function below these functions
```

```
have their trusted and untrusted properties defined in the EDL*/

extern "C" void ecall_set_key(const char *key, size_t len)
{
      if (len > 256)
            len = 256;
      memcpy(global_key, key, len);
      global_key_len = len;
}

extern "C" void ecall_rc4_sbox_init()
{
      rc4_init(s, global_key, global_key_len);
}

extern "C" void ecall_rc4_generate_keystream(size_t length)
{
      unsigned char s_local[256];
      memcpy(s_local, s, 256); // Use a local copy of S-box for keystream
    generation

      unsigned char data[256] = {0};
      unsigned char keystream[256]; // Local keystream buffer

      rc4_crypt(s_local, data, length, keystream);
      ocall_print_result((char *)keystream, length);
}

extern "C" void ecall_rc4_decrypt(char *ciphertext, size_t len)
{
      unsigned char s_local[256];
      memcpy(s_local, s, 256); // Use a local copy of S-box for decryption

      rc4_crypt(s_local, (unsigned char *)ciphertext, len, NULL);
      ocall_print_result(ciphertext, len);
}
```

Enclave.config.xml

```
<EnclaveConfiguration>
  <ProdID>0</ProdID>
  <ISVSVN>0</ISVSVN>
  <StackMaxSize>0x40000</StackMaxSize>
  <HeapMaxSize>0x100000</HeapMaxSize>
  <TCSNum>10</TCSNum>
  <TCSPolicy>1</TCSPolicy>
  <!-- Recommend changing 'DisableDebug' to 1 to make the enclave undebuggable for
    enclave release -->
  <DisableDebug>0</DisableDebug>
  <MiscSelect>0</MiscSelect>
  <MiscMask>0xFFFFFFFF</MiscMask>
</EnclaveConfiguration>
```