CS3312 Lab Report SGX应用开发

在Intel SGX Enclave内实现RC4算法

Osamu Takenaka 520030990026

2. 实验环境

2.1 服务器配置

- **型号**：ECS.g7.xlarge
- **CPU**：4核 (vCPU)
- **内存**：16 GB
- **硬盘类型**：高效云盘
- **硬盘大小**：40 GB

2.2 操作系统

- **操作系统类型**：Ubuntu
- **版本**：20.04 LTS (Focal Fossa)
- **架构**：64位
- **内核版本**：5.4.0-42-generic

2.3 开发环境

- **Intel SGX SDK**：
  - **版本**：2.12
  - **编译环境**：GCC 9.3.0
  - **链接器**：GNU ld (GNU Binutils for Ubuntu) 2.34
- **IDE**：
  - **使用IDE**：Visual Studio Code
  - **版本**：1.46.0

2.4 网络配置

- **VPC ID**：vpc-j6cyai6taix0ms47npsp
- **带宽**：5 Mbps
- **网络类型**：经典网络

3. 实验步骤和实现

在开源项目 https://github.com/digawp/hello-enclave.git 的基础上，实现了RC4算法的加密和解密功能。

本项目仓库地址：https://github.com/OsamuSkyhacker/CS3312-SGX-RC4

3.1 代码结构

项目主要文件：

- `App.cpp`：主应用程序，负责用户交互和调用Enclave函数。
- `Enclave.cpp`：Enclave内的实现文件，包含RC4算法的加密和解密逻辑。
- `Enclave.edl`：Enclave定义文件，定义了Enclave与应用之间的接口。

App.cpp文件的主要代码如下：

```
#include <stdio.h>
#include <iostream>
#include <iomanip>  // 用于 std::hex 和 std::setw
#include <string>   // 用于 std::string 和 std::getline
#include "Enclave_u.h"
#include "sgx_urts.h"
#include "sgx_utils/sgx_utils.h"

/* 全局 EID，由多个线程共享 */
sgx_enclave_id_t global_eid = 0;

// OCall 实现
```

```cpp
void ocall_print(const char* str) {
    printf("%s\n", str);
}

void print_hex(const char* title, const unsigned char* data, size_t data_len) {
    std::cout << title;
    for (size_t i = 0; i < data_len; ++i) {
        printf("%02x ", data[i]);
    }
    std::cout << std::endl;
}

int main(int argc, char const *argv[]) {
    if (initialize_enclave(&global_eid, "enclave.token", "enclave.signed.so") < 0) {
        std::cout << "初始化 Enclave 失败。" << std::endl;
        return 1;
    }

    std::string input_string;
    std::cout << "请输入要加密的字符串: ";
    std::getline(std::cin, input_string);  // 从标准输入读取一行

    const unsigned char rc4_key[] = "my_secret_key";
    sgx_status_t status = rc4_init(global_eid, rc4_key, sizeof(rc4_key) - 1);
    if (status != SGX_SUCCESS) {
        std::cout << "RC4 密钥初始化失败。" << std::endl;
        return 1;
    }

    unsigned char* data = new unsigned char[input_string.size() + 1];
    memcpy(data, input_string.c_str(), input_string.size() + 1);
    size_t data_len = input_string.size();

    std::cout << "原始数据: " << data << std::endl;
    print_hex("原始数据(HEX): ", data, data_len);

    // 加密数据
    status = rc4_crypt(global_eid, data, data_len);
    if (status != SGX_SUCCESS) {
        std::cout << "加密失败。" << std::endl;
        delete[] data;
        return 1;
    }
    std::cout << std::endl;
    print_hex("加密数据(HEX): ", data, data_len);
    std::cout << std::endl;


    // 重新初始化 RC4 状态
    rc4_init(global_eid, rc4_key, sizeof(rc4_key) - 1);

    // 解密数据
    status = rc4_crypt(global_eid, data, data_len);
    if (status != SGX_SUCCESS) {
        std::cout << "解密失败。" << std::endl;
        delete[] data;
        return 1;
    }

    std::cout << "解密数据: " << data << std::endl;
    print_hex("解密数据(HEX): ", data, data_len);

    delete[] data;  // 不要忘记释放内存
    return 0;
}
```

Enclave.cpp文件的主要代码如下：

```cpp
#include "Enclave_t.h"
#include <string.h> // for memcpy


int secret_print_helloworld() {
    ocall_print("I Love SJTU");
    return 1896;
```

```
}


// RC4 状态结构
struct rc4_state {
    unsigned char S[256];
    int i, j;
};

static rc4_state state;

// KSA: 密钥调度算法
void rc4_init(const unsigned char *key, size_t len) {
    int i, j;
    unsigned char t;

    for (i = 0; i < 256; i++) {
        state.S[i] = (unsigned char)i;
    }

    for (i = 0, j = 0; i < 256; i++) {
        j = (j + state.S[i] + key[i % len]) % 256;
        t = state.S[i];
        state.S[i] = state.S[j];
        state.S[j] = t;
    }

    state.i = 0;
    state.j = 0;
}

// PRGA: 伪随机生成算法
void rc4_crypt(unsigned char *data, size_t len) {
    int i = state.i, j = state.j;
    unsigned char t;

    for (size_t k = 0; k < len; k++) {
        i = (i + 1) % 256;
        j = (j + state.S[i]) % 256;
        t = state.S[i];
        state.S[i] = state.S[j];
        state.S[j] = t;
        data[k] ^= state.S[(state.S[i] + state.S[j]) % 256];
    }

    state.i = i;
    state.j = j;
}
```

Enclave.edl文件的主要代码如下：

```
enclave {
    from "Sealing/Sealing.edl" import *;

    trusted {
        /* define ECALLs here. */
        public int secret_print_helloworld(void);
        public void rc4_init([in, size=len] const unsigned char* key, size_t len);
        public void rc4_crypt([in,out, size=len] unsigned char* data, size_t len);
    };


    untrusted {
        /* define OCALLs here. */
        void ocall_print([in, string]const char* str);
    };
};
```

4. 实验结果

```
root@iZj6cej4ju4e8zfvifxrvfZ:~/hello-enclave# make all
GEN  =>  App/Enclave_u.c
CC   <=  App/Enclave_u.c
CXX  <=  App/App.cpp
```

```
CXX  <=  App/sgx_utils/sgx_utils.cpp
LINK =>  app
GEN  =>  Enclave/Enclave_t.c
CC   <=  Enclave/Enclave_t.c
CXX  <=  Enclave/Enclave.cpp
CXX  <=  Enclave/Sealing/Sealing.cpp
LINK =>  enclave.so
<!-- Please refer to User's Guide for the explanation of each field -->
<EnclaveConfiguration>
    <ProdID>0</ProdID>
    <ISVSVN>0</ISVSVN>
    <StackMaxSize>0x40000</StackMaxSize>
    <HeapMaxSize>0x100000</HeapMaxSize>
    <TCSNum>10</TCSNum>
    <TCSPolicy>1</TCSPolicy>
    <DisableDebug>0</DisableDebug>
    <MiscSelect>0</MiscSelect>
    <MiscMask>0xFFFFFFFF</MiscMask>
</EnclaveConfiguration>
tcs_num 10, tcs_max_num 10, tcs_min_pool 1
INFO: Enclave configuration 'MiscSelect' and 'MiscSelectMask' will prevent enclave from using dynamic
features. To use the dynamic features on SGX2 platform, suggest to set MiscMask[0]=0 and
MiscSelect[0]=1.
The required memory is 4968448B.
The required memory is 0x4bd000, 4852 KB.
handle_compatible_metadata: Overwrite with metadata version 0x100000005
Succeed.
SIGN =>  enclave.signed.so
```

编译完成后，运行 `./app`，输入要加密的字符串，程序会对其进行RC4加密和解密。

```
./app
请输入要加密的字符串: Hello SJTU
原始数据: Hello SJTU
原始数据(HEX): 48 65 6c 6c 6f 20 53 4a 54 55

加密数据(HEX): 01 ef c7 28 52 dd 26 6b 42 e1

解密数据: Hello SJTU
解密数据(HEX): 48 65 6c 6c 6f 20 53 4a 54 55
```

5. 运行截图