SGX Sample Code 분석

한성대 김경호

https://youtu.be/IUz7oZyPb2I

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

1. SGX 파일 구조

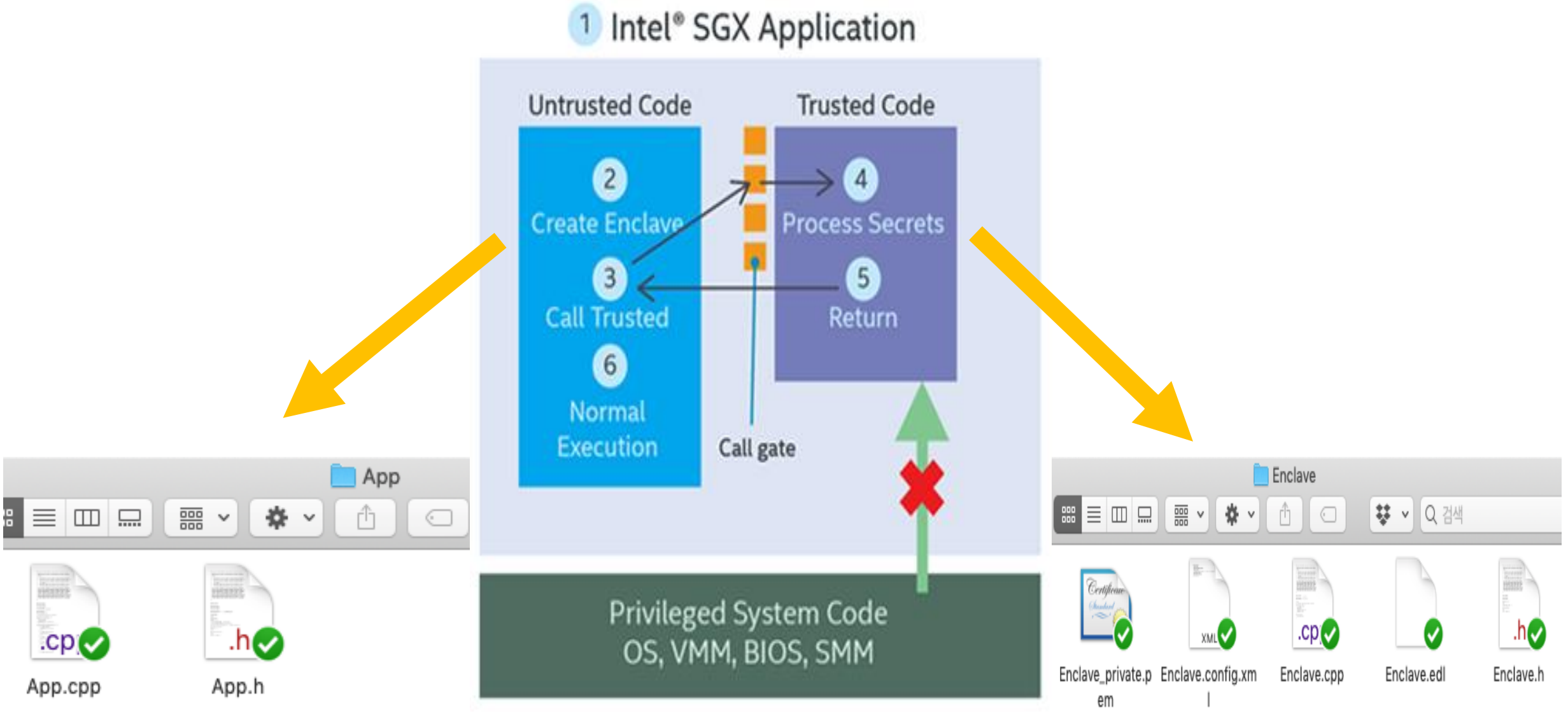
2. Password Wallet

3. Local Attestation

4. 향후 계획

CryptoCraft LAB

1. SGX 파일 구조


1 Intel® SGX Application
Untrusted Code
Trusted Code
2 Create Enclave
3 Call Trusted
4 Process Secrets
5 Return
6 Normal Execution
Call gate
Privileged System Code
OS, VMM, BIOS, SMM
App
App.cpp
App.h
Enclave
Enclave_private.pem
Enclave.config.xml
Enclave.cpp
Enclave.edl
Enclave.h

1. SGX 파일 구조 | Enclave.config.xml


```
<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>
```

- Enclave의 기본적인 파라미터 설정

ProdID : 제작자가 Enclave에 ID를 설정

ISVSVN : Enclave의 보안 버전 넘버
제작자가 보안 업데이트 있으면 버전을 증가

StackMaxSize : 쓰레드의 stack 크기

HeapMaxSize : 쓰레드의 Heap 크기

TCSNum : TCS의 최대 갯수

TCSPolicy : TCS 관리 Policy (Default 1)

DisableDebug : 1일 경우 디버깅 불가능

1. SGX 파일 구조 | Enclave_private.pem

enclave_private.pem

```
-----BEGIN RSA PRIVATE KEY-----
MIIG4gIBAAKCAYEAroOogvsj/fZDZY8XFdkl6dJmky0lRvnWMmpeH41Bla6U1qLZ
AmZuyIF+mQC/cgojIsrBMzBxb1kKqzATF4+XwPwgKz7fmiddmHyYz2WDJfAjIveJ
ZjdMjM4+EytGlkkJ52T8V8ds0/L2qKexJ+NBLxkeQLfV8n1mIk7zX7jguwbCG1Pr
nEMdJ3Sew20vnje+RsngAzdPChoJpVsWi/K7cettX/tbnre1DL02GXc5qJoQYk7b
3zkmhz31TgFrd9VVtmUGyFXAysuSAb3EN+5VnHGr0xKkeg8utErea2FNtNIgua8H
ONfm9Eiyaav1SVKzPHlyqLtcdxH3I8Wg7yqMsaprZ1n5A1v/levxnL8+It02KseD
5HqV4rf/cImSlCt3lpRg8U5E1pyFQ2IVEC/XTDMiI3c+AR+w2jSRB3Bwn9zJtFlW
KHG3m1xGI4ck+Lci1JvWWLXQagQSPtZTsubxTQNx1gsgZhgv1JHVZMdbVlAbbRMC
1nSuJNl7KPAS/VfzAgEDAoIBgHRXxaynbVP5gkO0ug6Qw/E27wzIw4SmjsxG6Wpe
K7kfDeRskKxESdsA/xCrKkwGwhcx1iIgS5+Qscd1Yg+1D9X9asd/P7waPmWoZd+Z
AhlKwhdPsO7PiF3e1AzHhGQwsUTt/Y/aSI1MpHBvy2/s1h9mFCslOUxTmWw0oj/Q
ldIEgWeNR72CE2+jFIJIyml6ftnb6qzPiga8Bm48ubKh0kvySOqnkmnPzgh+JBD6
JnBmtZbfPT97bwTT+N6rnPqOOApvfHPf15kWI8yDbprG1l4OCUaIUH1AszxLd826
5IPM+8gINLRDP1MA6azECPjTyHXhtnSIBZCyWSVkc05vYmNXYUNiXWMajcxW9M02
wKzFELO8NCEAkaTPxwo4SCyIjUxiK1LbQ9h8PSy4c1+gGP4LAMR8xqP4QKg6zdu9
osUGG/xRe/uufgTBFkcjqBHtK5L5VI0jeNIUAgW/6iNbYXjBMJ0GfauLs+g1VsOm
WfdgXzsb9DYdMa0OXXHypmV4GwKBwQDUwQj8RKJ6c8cT4vcWCoJvJF00+RFL+P3i
Gx2DLERxRrDa8AVGfqaCjsR+3vLgG8V/py+z+dxZYSqeB80Qeo6PDITcRKoeAYh9
xlT3LJOS+k1cJcEmlbbO2IjLkTmzSwa80fWexKu8/Xv6vv15gpqYl1ngYoqJM3pd
vzmTIOi7MKSZ0WmEQavrZj8zK4endE3v0eAEeQ55j1GImbypSf7Idh7wOXtjZ7WD
Dg6yWDrri+AP/L3gClMj8wsAxMV4ZR8CgcEA0fzDHkFa6raVOxWnObmRoDhAtE0a
cjUj976NM5yyfdf2MrKy4/RhdTiPZ6b08/lBC/+xRfV3xKVGzacm6QjqjZrUpgHC
0LKiZaMtccCJjLtPwQd0jGQEnKfMFaPsnhOc5y8qVkCzVOSthY5qhz0XNotHHFmJ
gffVgB0iqrMTvSL7IA2yqqpOqNRlhaYhNl8TiFP3gIeMtVa9rZy31JPgT2uJ+kfo
gV7sdTPEjPWZd7OshGxWpT6QfVDj/T9T7L6tAoHBAI3WBf2DFvxNL2KXT2QHAZ9t
k3imC4f7U+wSE6zILaDZyzygA4RUbwG0gv8/TJVn2P/Eynf76DuWHGlaiLWnCbSz
Az2DHBQBBaku409zDQym3j1ugMRjzzSQWzJg0SIyBH3hTmnYcn3+Uqcp/lEBvGW6
O+rsXFt3pukqJmIV8HzLGGaLm62BHUeZf3dyWm+i3p/hQAL7Xvu04QW70xuGqdr5
afV7p5eaeQIJXyGQJ0eylV/90+qxjMKiB1XYg6WYvwKBwQCL/ddpgOdHJGN8uRom
e7Zq0Csi3hGheMKlKbN3vcxT5U7MdyHtTZZOJbTvxKNNUNYH/8uD+PqDGNneb29G
BfGzvI3EASyLIcGZF3OhKwZd0jUrWk2y7Vhob91jwp2+t73vdMbkKyI4mHOuXvGv
fg95si9oO7EBT+Oqvhccd2J+F1IVXncccYnF4u5ZGWt5lLewN/pVr7MjjykeaHqN
t+rfnQam2psA6fL4zS2zTmZPzR2tnY8Y1GBTi0Ko1OKd1HMCgcAb5cB/7/AQlhP9
yQa04PLH9ygQkKKptZp7dy5WcWRx0K/hAHRoi2aw1wZqfm7VBNu2SLcs90kCCCxp
6C5sfJi6b8NpNbIPC+sc9wsFr7pGo9SFzQ78UlcWYK2Gu2FxlMjonhka5hvo4zvg
WxlpXKEkaFt3gLd92m/dMqBrHfafH7VwOJY2zT3WIpjwuk0ZzmRg5p0pG/svVQEH
NZmwRwlopysbR69B/n1nefJ84UO50fLh5s5Zr3gBRwbWNZyzhXk=
-----END RSA PRIVATE KEY-----
```

- Enclave 서명에 필요한 Private Key
- SGX는 3072bit RSA, Exponent는 3 사용
- 해당 명령어를 사용하면 위의 형식의 Private Key 파일 생성 가능

Openssl genrsa –out Enclave_private.pem -3 3072

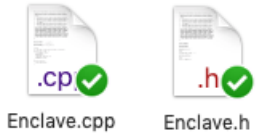
1. SGX 파일 구조 | Enclave.edl

Enclave.edl

```
enclave {

    /* Import ECALL/OCALL from sub-directory EDLs.
     *  [from]: specifies the location of EDL file.
     *  [import]: specifies the functions to import,
     *  [*]: implies to import all functions.
     */

    trusted {
        public void printf_helloworld();
    };

    /*
     * ocall_print_string - invokes OCALL to display string buffer inside the enclave.
     *  [in]: copy the string buffer to App outside.
     *  [string]: specifies 'str' is a NULL terminated buffer.
     */
    untrusted {
        void ocall_print_string([in, string] const char *str);
    };

};
```

- Enclave 내부에서 사용할 함수와 외부 응용 프로그램에서 사용될 함수를 분리하여 작성
- 앱 제작시 항상 Trust, Untrust를 구분하여 작성해야만 컴파일이 가능

1. SGX 파일 구조 | Enclave.cpp & Enclave.h



- Trust 환경에서 실행되는 함수 정의

```c
#include <stdarg.h>
#include <stdio.h>      /* vsnprintf */

#include "Enclave.h"
#include "Enclave_t.h"  /* print_string */

/*
 * printf:
 *   Invokes OCALL to display the enclave buffer to the terminal.
 */
void printf(const char *fmt, ...)
{
    char buf[BUFSIZ] = {'\0'};
    va_list ap;
    va_start(ap, fmt);
    vsnprintf(buf, BUFSIZ, fmt, ap);
    va_end(ap);
    ocall_print_string(buf);
}

void printf_helloworld()
{
    printf("Hello World\n");
}
```

```c
#ifndef _ENCLAVE_H_
#define _ENCLAVE_H_

#include <stdlib.h>
#include <assert.h>

#if defined(__cplusplus)
extern "C" {
#endif

void printf(const char *fmt, ...);
void printf_helloworld();

#if defined(__cplusplus)
}
#endif

#endif /* !_ENCLAVE_H_ */
```

2. Password Wallet | 파일 구조

2. Password Wallet | app.cpp (1 / 2)

- Enclave 생성

- 실행파일 실행 시 입력된 파라미터를 파싱하여 switch / case 문을 이용하여 분류

```
enclave_status = sgx_create_enclave(ENCLAVE_FILE, SGX_DEBUG_FLAG, &token, &updated, &eid, NULL);
if(enclave_status != SGX_SUCCESS) {
    error_print("Fail to initialize enclave.");
    return -1;
}
info_print("Enclave successfully initilised.");
```

```
const char* options = "hvn:p:c:sax:y:z:r:";
opterr=0; // prevent 'getopt' from printing err messages
char err_message[100];
int opt, stop=0;
int h_flag=0, v_flag=0, s_flag=0, a_flag=0;
char * n_value=NULL, *p_value=NULL, *c_value=NULL, *x_value=NULL, *y_value=NULL, *z_value=NULL, *r_value=NULL;

// read user input
while ((opt = getopt(argc, argv, options)) != -1) {
    switch (opt) {
        // help
        case 'h':
            h_flag = 1;
            break;

        // create new wallet
        case 'n':
            n_value = optarg;
            break;

        // master-password
        case 'p':
            p_value = optarg;
            break;
```

2. Password Wallet | app.cpp (2 / 2)

- 각 파라미터 값에 따라 원하는 연산을 실행
- 생성, 변경, 추가, 제거
- 앱 종료시 만들어진 Enclave 제거

```
else if(n_value!=NULL) {
    ecall_status = ecall_create_wallet(eid, &ret, n_value);
    if (ecall_status != SGX_SUCCESS || is_error(ret)) {
        error_print("Fail to create new wallet.");
    }
    else {
        info_print("Wallet successfully created.");
    }
}

// change master-password
else if (p_value!=NULL && c_value!=NULL) {
    ecall_status = ecall_change_master_password(eid, &ret, p_value, c_value);
    if (ecall_status != SGX_SUCCESS || is_error(ret)) {
        error_print("Fail change master-password.");
    }
    else {
        info_print("Master-password successfully changed.");
    }
}

// show wallet
else if(p_value!=NULL && s_flag) {
    wallet_t* wallet = (wallet_t*)malloc(sizeof(wallet_t));
    ecall_status = ecall_show_wallet(eid, &ret, p_value, wallet, sizeof(wallet_t));
    if (ecall_status != SGX_SUCCESS || is_error(ret)) {
        error_print("Fail to retrieve wallet.");
    }
    else {
        info_print("Wallet successfully retrieved.");
        print_wallet(wallet);
    }
    free(wallet);
}

// add item
else if (p_value!=NULL && a_flag && x_value!=NULL && y_value!=NULL && z_value!=NULL) {
    item_t* new_item = (item_t*)malloc(sizeof(item_t));
    strcpy(new_item->title, x_value);
    strcpy(new_item->username, y_value);
    strcpy(new_item->password, z_value);
    ecall_status = ecall_add_item(eid, &ret, p_value, new_item, sizeof(item_t));
    if (ecall_status != SGX_SUCCESS || is_error(ret)) {
        error_print("Fail to add new item to wallet.");
    }
    else {
        info_print("Item successfully added to the wallet.");
    }
    free(new_item);
}
```

```
enclave_status = sgx_destroy_enclave(eid);
if(enclave_status != SGX_SUCCESS) {
    error_print("Fail to destroy enclave.");
    return -1;
}
info_print("Enclave successfully destroyed.");
```

2. Password Wallet | enclave.cpp (1 / 5)

```
int ecall_create_wallet(const char* master_password) {

    sgx_status_t ocall_status, sealing_status;
    int ocall_ret;


    // 1. check passaword policy
    if (strlen(master_password) < 8 || strlen(master_password)+1 > MAX_ITEM_SIZE) {
        return ERR_PASSWORD_OUT_OF_RANGE;
    }


    // 2. abort if wallet already exist
    ocall_status = ocall_is_wallet(&ocall_ret);
    if (ocall_ret != 0) {
        return ERR_WALLET_ALREADY_EXISTS;
    }


    // 3. create new wallet
    wallet_t* wallet = (wallet_t*)malloc(sizeof(wallet_t));
    wallet->size = 0;
    strncpy(wallet->master_password, master_password, strlen(master_password)+1);


    // 4. seal wallet
    size_t sealed_size = sizeof(sgx_sealed_data_t) + sizeof(wallet_t);
    uint8_t* sealed_data = (uint8_t*)malloc(sealed_size);
    sealing_status = seal_wallet(wallet, (sgx_sealed_data_t*)sealed_data, sealed_size);
    free(wallet);
    if (sealing_status != SGX_SUCCESS) {
        free(sealed_data);
        return ERR_FAIL_SEAL;
    }


    // 5. save wallet
    ocall_status = ocall_save_wallet(&ocall_ret, sealed_data, sealed_size);
    free(sealed_data);
    if (ocall_ret != 0 || ocall_status != SGX_SUCCESS) {
        return ERR_CANNOT_SAVE_WALLET;
    }


    // 6. exit enclave
    return RET_SUCCESS;
}
```

Wallet 생성 ECALL

- 입력된 비밀번호 체크
- 이미 Wallet이 있는지 확인
- 새로운 Wallet 생성
- Wallet을 Sealing
- Sealed 된 Wallet 저장

2. Password Wallet | enclave.cpp (2 / 5)

```
int ecall_show_wallet(const char* master_password, wallet_t* wallet, size_t wallet_size) {

    sgx_status_t ocall_status, sealing_status;
    int ocall_ret;



    // 1. load wallet
    size_t sealed_size = sizeof(sgx_sealed_data_t) + sizeof(wallet_t);
    uint8_t* sealed_data = (uint8_t*)malloc(sealed_size);
    ocall_status = ocall_load_wallet(&ocall_ret, sealed_data, sealed_size);
    if (ocall_ret != 0 || ocall_status != SGX_SUCCESS) {
        free(sealed_data);
        return ERR_CANNOT_LOAD_WALLET;
    }


    // 2. unseal loaded wallet
    uint32_t plaintext_size = sizeof(wallet_t);
    wallet_t* unsealed_wallet = (wallet_t*)malloc(plaintext_size);
    sealing_status = unseal_wallet((sgx_sealed_data_t*)sealed_data, unsealed_wallet, plaintext_size);
    free(sealed_data);
    if (sealing_status != SGX_SUCCESS) {
        free(unsealed_wallet);
        return ERR_FAIL_UNSEAL;
    }


    // 3. verify master-password
    if (strcmp(unsealed_wallet->master_password, master_password) != 0) {
        free(unsealed_wallet);
        return ERR_WRONG_MASTER_PASSWORD;
    }


    // 4. return wallet to app
    (* wallet) = *unsealed_wallet;
    free(unsealed_wallet);


    // 5. exit enclave
    return RET_SUCCESS;
}
```

Wallet 출력 ECALL

- 저장된 Wallet 로드
- Wallet을 Unsealing
- 입력된 Password 확인
- Unsealing된 값을 리턴값에 저장 후 free

2. Password Wallet | enclave.cpp (3 / 5)

```
int ecall_change_master_password(const char* old_password, const char* new_password) {

    sgx_status_t ocall_status, sealing_status;
    int ocall_ret;

    // 1. check passaword policy
    if (strlen(new_password) < 8 || strlen(new_password)+1 > MAX_ITEM_SIZE) {
        return ERR_PASSWORD_OUT_OF_RANGE;
    }

    // 2. load wallet
    size_t sealed_size = sizeof(sgx_sealed_data_t) + sizeof(wallet_t);
    uint8_t* sealed_data = (uint8_t*)malloc(sealed_size);
    ocall_status = ocall_load_wallet(&ocall_ret, sealed_data, sealed_size);
    if (ocall_ret != 0 || ocall_status != SGX_SUCCESS) {
        free(sealed_data);
        return ERR_CANNOT_LOAD_WALLET;
    }

    // 3. unseal wallet
    uint32_t plaintext_size = sizeof(wallet_t);
    wallet_t* wallet = (wallet_t*)malloc(plaintext_size);
    sealing_status = unseal_wallet((sgx_sealed_data_t*)sealed_data, wallet, plaintext_size);
    free(sealed_data);
    if (sealing_status != SGX_SUCCESS) {
        free(wallet);
        return ERR_FAIL_UNSEAL;
    }


    // 4. verify master-password
    if (strcmp(wallet->master_password, old_password) != 0) {
        free(wallet);
        return ERR_WRONG_MASTER_PASSWORD;
    }

    // 5. update password
    strncpy(wallet->master_password, new_password, strlen(new_password)+1);


    // 6. seal wallet
    sealed_data = (uint8_t*)malloc(sealed_size);
    sealing_status = seal_wallet(wallet, (sgx_sealed_data_t*)sealed_data, sealed_size);
    free(wallet);
    if (sealing_status != SGX_SUCCESS) {
        free(wallet);
        free(sealed_data);
        return ERR_FAIL_SEAL;
    }


    // 7. save wallet
    ocall_status = ocall_save_wallet(&ocall_ret, sealed_data, sealed_size);
    free(sealed_data);
    if (ocall_ret != 0 || ocall_status != SGX_SUCCESS) {
        return ERR_CANNOT_SAVE_WALLET;
    }
```

Wallet Password 변경 ECALL

- 입력된 비밀번호 제약 조건 체크
- 저장된 Wallet 로드
- Wallet을 Unsealing
- 입력된 Password 확인
- 새로 입력된 Password로 교환
- 저장된 값 Sealing
- Sealed 된 Wallet 저장

2. Password Wallet | enclave.cpp (4 / 5)

```
int ecall_add_item(const char* master_password, const item_t* item, const size_t item_size) {

    sgx_status_t ocall_status, sealing_status;
    int ocall_ret;

    // 2. load wallet
    size_t sealed_size = sizeof(sgx_sealed_data_t) + sizeof(wallet_t);
    uint8_t* sealed_data = (uint8_t*)malloc(sealed_size);
    ocall_status = ocall_load_wallet(&ocall_ret, sealed_data, sealed_size);
    if (ocall_ret != 0 || ocall_status != SGX_SUCCESS) {
        free(sealed_data);
        return ERR_CANNOT_LOAD_WALLET;
    }

    // 3. unseal wallet
    uint32_t plaintext_size = sizeof(wallet_t);
    wallet_t* wallet = (wallet_t*)malloc(plaintext_size);
    sealing_status = unseal_wallet((sgx_sealed_data_t*)sealed_data, wallet, plaintext_size);
    free(sealed_data);
    if (sealing_status != SGX_SUCCESS) {
        free(wallet);
        return ERR_FAIL_UNSEAL;
    }

    // 3. verify master-password
    if (strcmp(wallet->master_password, master_password) != 0) {
        free(wallet);
        return ERR_WRONG_MASTER_PASSWORD;
    }

    // 4. check input length
    if (strlen(item->title)+1 > MAX_ITEM_SIZE ||
        strlen(item->username)+1 > MAX_ITEM_SIZE ||
        strlen(item->password)+1 > MAX_ITEM_SIZE
    ) {
        free(wallet);
        return ERR_ITEM_TOO_LONG;
    }

    // 5. add item to the wallet
    size_t wallet_size = wallet->size;
    if (wallet_size >= MAX_ITEMS) {
        free(wallet);
        return ERR_WALLET_FULL;
    }
    wallet->items[wallet_size] = *item;
    ++wallet->size;
    // 6. seal wallet
    sealed_data = (uint8_t*)malloc(sealed_size);
    sealing_status = seal_wallet(wallet, (sgx_sealed_data_t*)sealed_data, sealed_size);
    free(wallet);
    if (sealing_status != SGX_SUCCESS) {
        free(wallet);
        free(sealed_data);
        return ERR_FAIL_SEAL;
    }

    // 7. save wallet
    ocall_status = ocall_save_wallet(&ocall_ret, sealed_data, sealed_size);
    free(sealed_data);
    if (ocall_ret != 0 || ocall_status != SGX_SUCCESS) {
        return ERR_CANNOT_SAVE_WALLET;
    }

    // 8. exit enclave
    return RET_SUCCESS;
}
```

Wallet item 추가 ECALL

- 저장된 Wallet 로드
- Wallet을 Unsealing
- 입력된 Password 확인
- 입력된 아이템 제약 조건 확인 후 저장
- 저장된 값 Sealing
- Sealed 된 Wallet 저장

2. Password Wallet | enclave.cpp (5 / 5)

```
int ecall_remove_item(const char* master_password, const int index) {

    sgx_status_t ocall_status, sealing_status;
    int ocall_ret;

    // 1. check index bounds
    if (index < 0 || index >= MAX_ITEMS) {
        return ERR_ITEM_DOES_NOT_EXIST;
    }

    // 2. load wallet
    size_t sealed_size = sizeof(sgx_sealed_data_t) + sizeof(wallet_t);
    uint8_t* sealed_data = (uint8_t*)malloc(sealed_size);
    ocall_status = ocall_load_wallet(&ocall_ret, sealed_data, sealed_size);
    if (ocall_ret != 0 || ocall_status != SGX_SUCCESS) {
        free(sealed_data);
        return ERR_CANNOT_LOAD_WALLET;
    }

    // 3. unseal wallet
    uint32_t plaintext_size = sizeof(wallet_t);
    wallet_t* wallet = (wallet_t*)malloc(plaintext_size);
    sealing_status = unseal_wallet((sgx_sealed_data_t*)sealed_data, wallet, plaintext_size);
    free(sealed_data);
    if (sealing_status != SGX_SUCCESS) {
        free(wallet);
        return ERR_FAIL_UNSEAL;
    }

    // 4. verify master-password
    if (strcmp(wallet->master_password, master_password) != 0) {
        free(wallet);
        return ERR_WRONG_MASTER_PASSWORD;
    }

    // 5. remove item from the wallet
    size_t wallet_size = wallet->size;
    if (index >= wallet_size) {
        free(wallet);
        return ERR_ITEM_DOES_NOT_EXIST;
    }
    for (int i = index; i < wallet_size-1; ++i) {
        wallet->items[i] = wallet->items[i+1];
    }
    --wallet->size;

    // 6. seal wallet
    sealed_data = (uint8_t*)malloc(sealed_size);
    sealing_status = seal_wallet(wallet, (sgx_sealed_data_t*)sealed_data, sealed_size);
    free(wallet);
    if (sealing_status != SGX_SUCCESS) {
        free(sealed_data);
        return ERR_FAIL_SEAL;
    }

    // 7. save wallet
    ocall_status = ocall_save_wallet(&ocall_ret, sealed_data, sealed_size);
    free(sealed_data);
    if (ocall_ret != 0 || ocall_status != SGX_SUCCESS) {
        return ERR_CANNOT_SAVE_WALLET;
```

Wallet item 제거 ECALL

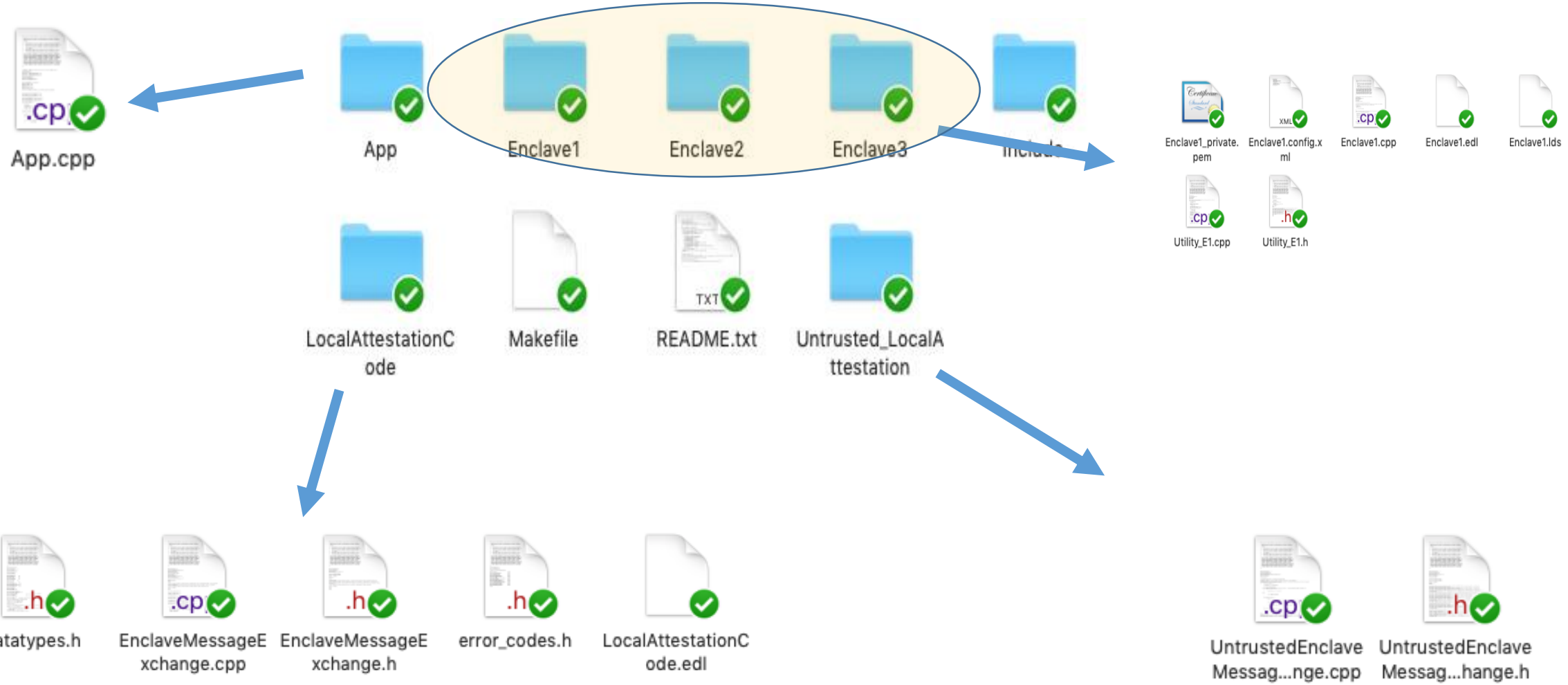
- Index 확인
- 저장된 Wallet 로드
- Wallet을 Unsealing
- 입력된 Password 확인
- 저장된 item 삭제
- 나머지 값 Sealing
- Sealed 된 Wallet 저장

2. Password Wallet | sealing.cpp

- Intel에서 제작한 API를 알맞게 호출하는 함수
- Sealing을 이용할 때 다음과 같은 방식으로 함수를 제작하여 사용하면 됨

```
#include "enclave_t.h"
#include "sgx_trts.h"
#include "sgx_tseal.h"

#include "wallet.h"
#include "sealing.h"

sgx_status_t seal_wallet(const wallet_t* wallet, sgx_sealed_data_t* sealed_data, size_t sealed_size) {
    return sgx_seal_data(0, NULL, sizeof(wallet_t), (uint8_t*)wallet, sealed_size, sealed_data);
}

sgx_status_t unseal_wallet(const sgx_sealed_data_t* sealed_data, wallet_t* plaintext, uint32_t plaintext_size) {
    return sgx_unseal_data(sealed_data, NULL, NULL, (uint8_t*)plaintext, &plaintext_size);
}
```

3. LocalAttestaion | 파일 구조



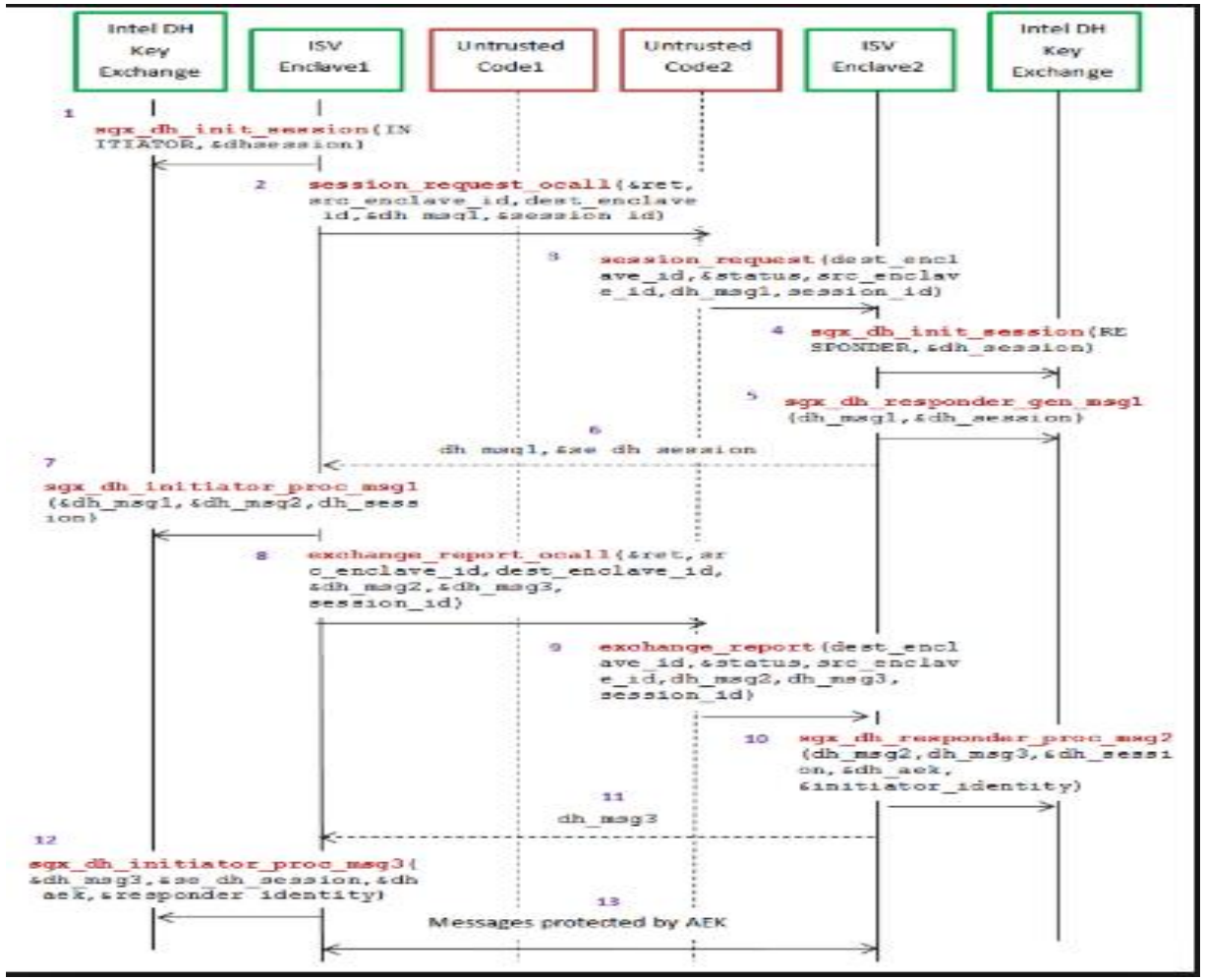
3. LocalAttestaion | App.cpp (1 / 2)

```
if(load_enclaves() != SGX_SUCCESS)
{
     printf("\nLoad Enclave Failure");
}

printf("\nAvailable Enclaves");
printf("\nEnclave1 - EnclaveID %" PRIx64, e1_enclave_id);
printf("\nEnclave2 - EnclaveID %" PRIx64, e2_enclave_id);
printf("\nEnclave3 - EnclaveID %" PRIx64, e3_enclave_id);
```

```
//Test Create session between Enclave1(Source) and Enclave2(Destination)
status = Enclave1_test_create_session(e1_enclave_id, &ret_status, e1_enclave_id, e2_enclave_id);
if (status!=SGX_SUCCESS)
{
    printf("Enclave1_test_create_session Ecall failed: Error code is %x", status);
    break;
}
else
{
    if(ret_status==0)
    {
        printf("\n\nSecure Channel Establishment between Source (E1) and Destination (E2) Enclaves successful !!!");
    }
    else
    {
        printf("\nSession establishment and key exchange failure between Source (E1) and Destination (E2): Error code is %x", ret_status);
        break;
    }
}

//Test Enclave to Enclave call between Enclave1(Source) and Enclave2(Destination)
status = Enclave1_test_enclave_to_enclave_call(e1_enclave_id, &ret_status, e1_enclave_id, e2_enclave_id);
if (status!=SGX_SUCCESS)
{
    printf("Enclave1_test_enclave_to_enclave_call Ecall failed: Error code is %x", status);
    break;
}
else
{
    if(ret_status==0)
    {
        printf("\n\nEnclave to Enclave Call between Source (E1) and Destination (E2) Enclaves successful !!!");
    }
    else
    {
        printf("\n\nEnclave to Enclave Call failure between Source (E1) and Destination (E2): Error code is %x", ret_status);
        break;
    }
}
```

```
    status = Enclave1_test_message_exchange(e1_enclave_id, &ret_status, e1_enclave_id, e2_enclave_id);
    if (status!=SGX_SUCCESS)
    {
        printf("Enclave1_test_message_exchange Ecall failed: Error code is %x", status);
        break;
    }
    else
    {
        if(ret_status==0)
        {
            printf("\n\nMessage Exchange between Source (E1) and Destination (E2) Enclaves successful !!!");
        }
        else
        {
            printf("\n\nMessage Exchange failure between Source (E1) and Destination (E2): Error code is %x", ret_status);
            break;
        }
    }
    //Test Enclave to Enclave call between Enclave1(Source) and Enclave3(Destination)

    //Test Closing Session between Enclave1(Source) and Enclave2(Destination)
    status = Enclave1_test_close_session(e1_enclave_id, &ret_status, e1_enclave_id, e2_enclave_id);
    if (status!=SGX_SUCCESS)
    {
        printf("Enclave1_test_close_session Ecall failed: Error code is %x", status);
        break;
    }
    else
    {
        if(ret_status==0)
        {
            printf("\n\nClose Session between Source (E1) and Destination (E2) Enclaves successful !!!");
        }
        else
        {
            printf("\n\nClose session failure between Source (E1) and Destination (E2): Error code is %x", ret_status);
            break;
        }
    }
   }while(0);

sgx_destroy_enclave(e1_enclave_id);
waitForKeyPress();
```

3. LocalAttestaion | Attestation의 과정 설명




```
status = sgx_dh_init_session(SGX_DH_SESSION_INITIATOR, &sgx_dh_session);
if(SGX_SUCCESS != status)
{
        return status;
}

//Ocall to request for a session with the destination enclave and obtain session id and Message 1 if successful
status = session_request_ocall(&retstatus, src_enclave_id, dest_enclave_id, &dh_msg1, &session_id);
if (status == SGX_SUCCESS)
{
    if ((ATTESTATION_STATUS)retstatus != SUCCESS)
        return ((ATTESTATION_STATUS)retstatus);
}
else
{
    return ATTESTATION_SE_ERROR;
}
//Process the message 1 obtained from desination enclave and generate message 2
status = sgx_dh_initiator_proc_msg1(&dh_msg1, &dh_msg2, &sgx_dh_session);
if(SGX_SUCCESS != status)
{
     return status;
}

//Send Message 2 to Destination Enclave and get Message 3 in return
status = exchange_report_ocall(&retstatus, src_enclave_id, dest_enclave_id, &dh_msg2, &dh_msg3, session_id);
if (status == SGX_SUCCESS)
{
    if ((ATTESTATION_STATUS)retstatus != SUCCESS)
        return ((ATTESTATION_STATUS)retstatus);
}
else
{
    return ATTESTATION_SE_ERROR;
}

//Process Message 3 obtained from the destination enclave
status = sgx_dh_initiator_proc_msg3(&dh_msg3, &sgx_dh_session, &dh_aek, &responder_identity);
```

<EnclaveMessageExchange.cpp>

3. LocalAttestaion | Enclave.cpp (1 / 4)

- 같은 CPU상에서 돌고있는 2개의 Enclave 사이에 안전한 세션 생성
- Enclave의 id와 세션 정보를 Map으로 관리

```
uint32_t test_create_session(sgx_enclave_id_t src_enclave_id,
                             sgx_enclave_id_t dest_enclave_id)
{
    ATTESTATION_STATUS ke_status = SUCCESS;
    dh_session_t dest_session_info;

    //Core reference code function for creating a session
    ke_status = create_session(src_enclave_id, dest_enclave_id, &dest_session_info);

    //Insert the session information into the map under the corresponding destination enclave id
    if(ke_status == SUCCESS)
    {
        g_src_session_info_map.insert(std::pair<sgx_enclave_id_t, dh_session_t>(dest_enclave_id, dest_session_info));
    }
    memset(&dest_session_info, 0, sizeof(dh_session_t));
    return ke_status;
}
```

3. LocalAttestaion | Enclave.cpp (2 / 4)

- Enclave2의 foo1 함수 호출하기위한 메세지 마셜링
- 암호화 하여 전송 후 수신
- 수신한 정보 언마셜링하여 메세지 교환

```
uint32_t test_enclave_to_enclave_call(sgx_enclave_id_t src_enclave_id,
                                      sgx_enclave_id_t dest_enclave_id)
{
    ATTESTATION_STATUS ke_status = SUCCESS;
    uint32_t var1,var2;
    uint32_t target_fn_id, msg_type;
    char* marshalled_inp_buff;
    size_t marshalled_inp_buff_len;
    char* out_buff;
    size_t out_buff_len;
    dh_session_t *dest_session_info;
    size_t max_out_buff_size;
    char* retval;

    var1 = 0x4;
    var2 = 0x5;
    target_fn_id = 0;
    msg_type = ENCLAVE_TO_ENCLAVE_CALL;
    max_out_buff_size = 50;

    //Marshals the input parameters for calling function foo1 in Enclave2 into a input buffer
    ke_status = marshal_input_parameters_e2_foo1(target_fn_id, msg_type, var1, var2, &marshalled_inp_buff, &marshalled_inp_buff_len);
    if(ke_status != SUCCESS)
    {
        return ke_status;
    }

    //Search the map for the session information associated with the destination enclave id of Enclave2 passed in
    std::map<sgx_enclave_id_t, dh_session_t>::iterator it = g_src_session_info_map.find(dest_enclave_id);
    if(it != g_src_session_info_map.end())
    {
        dest_session_info = &it->second;
    }
    else
    {
        SAFE_FREE(marshalled_inp_buff);
        return INVALID_SESSION;
    }

    //Core Reference Code function
    ke_status = send_request_receive_response(src_enclave_id, dest_enclave_id, dest_session_info, marshalled_inp_buff,
                                              marshalled_inp_buff_len, max_out_buff_size, &out_buff, &out_buff_len);

    if(ke_status != SUCCESS)
    {
        SAFE_FREE(marshalled_inp_buff);
        SAFE_FREE(out_buff);
        return ke_status;
    }

    //Un-marshal the return value and output parameters from foo1 of Enclave 2
    ke_status = unmarshal_retval_and_output_parameters_e2_foo1(out_buff, &retVal);
    if(ke_status != SUCCESS)
    {
        SAFE_FREE(marshalled_inp_buff);
        SAFE_FREE(out_buff);
        return ke_status;
    }
```

3. LocalAttestaion | Enclave.cpp (3 / 4)

- Enclave2와 메세지 교환
- 앞의 Enclave to Enclave 함수와 매우 유사

```
uint32_t test_message_exchange(sgx_enclave_id_t src_enclave_id,
                               sgx_enclave_id_t dest_enclave_id)
{
    ATTESTATION_STATUS ke_status = SUCCESS;
    uint32_t target_fn_id, msg_type;
    char* marshalled_inp_buff;
    size_t marshalled_inp_buff_len;
    char* out_buff;
    size_t out_buff_len;
    dh_session_t *dest_session_info;
    size_t max_out_buff_size;
    char* secret_response;
    uint32_t secret_data;

    target_fn_id = 0;
    msg_type = MESSAGE_EXCHANGE;
    max_out_buff_size = 50;
    secret_data = 0x12345678; //Secret Data here is shown only for purpose of demonstration.

    //Marshals the secret data into a buffer
    ke_status = marshal_message_exchange_request(target_fn_id, msg_type, secret_data, &marshalled_inp_buff, &marshalled_inp_buff_len);
    if(ke_status != SUCCESS)
    {
        return ke_status;
    }
    //Search the map for the session information associated with the destination enclave id passed in
    std::map<sgx_enclave_id_t, dh_session_t>::iterator it = g_src_session_info_map.find(dest_enclave_id);
    if(it != g_src_session_info_map.end())
    {
         dest_session_info = &it->second;
    }
    else
    {
        SAFE_FREE(marshalled_inp_buff);
        return INVALID_SESSION;
    }

    //Core Reference Code function
    ke_status = send_request_receive_response(src_enclave_id, dest_enclave_id, dest_session_info, marshalled_inp_buff,
                                              marshalled_inp_buff_len, max_out_buff_size, &out_buff, &out_buff_len);
    if(ke_status != SUCCESS)
    {
        SAFE_FREE(marshalled_inp_buff);
        SAFE_FREE(out_buff);
        return ke_status;
    }

    //Un-marshal the secret response data
    ke_status = umarshal_message_exchange_response(out_buff, &secret_response);
    if(ke_status != SUCCESS)
    {
        SAFE_FREE(marshalled_inp_buff);
        SAFE_FREE(out_buff);
        return ke_status;
    }

    SAFE_FREE(marshalled_inp_buff);
    SAFE_FREE(out_buff);
    SAFE_FREE(secret_response);
    return SUCCESS;
```

3. LocalAttestaion | Enclave.cpp (4 / 4)

- Map에 저장된 모든 세션 관련 정보 삭제

```
uint32_t test_close_session(sgx_enclave_id_t src_enclave_id,
                            sgx_enclave_id_t dest_enclave_id)
{
    dh_session_t dest_session_info;
    ATTESTATION_STATUS ke_status = SUCCESS;
    //Search the map for the session information associated with the destination enclave id passed in
    std::map<sgx_enclave_id_t, dh_session_t>::iterator it = g_src_session_info_map.find(dest_enclave_id);
    if(it != g_src_session_info_map.end())
    {
        dest_session_info = it->second;
    }
    else
    {
        return NULL;
    }

    //Core reference code function for closing a session
    ke_status = close_session(src_enclave_id, dest_enclave_id);

    //Erase the session information associated with the destination enclave id
    g_src_session_info_map.erase(dest_enclave_id);
    return ke_status;
}
```

4. 향후 계획

- Remote Attestation 분석
- Makefile 분석
- Sealing , Attestation 을 이용하여 App 제작

Q & A