上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SGX 客户端开发实验报告

学号: 522031910439　姓名: 梁俊轩

2025 年 6 月 15 日

1　介绍

1.1　SGX

2013 年, Intel 推出 SGX(software guard extensions) 指令集扩展 [1], 如图1所示，旨在以硬件安全为强制性保障, 不依赖于固件和软件的安全状态, 提供用户空间的可信执行环境, 通过一组新的指令集扩展与访问控制机制, 实现不同程序间的隔离运行, 保障用户关键代码和数据的机密性与完整性不受恶意软件的破坏. 不同于其他安全技术, SGX 的可信计算基 (trusted computing base, 简称 TCB) 仅包括硬件, 避免了基于软件的 TCB 自身存在软件安全漏洞与威胁的缺陷, 极大地提升了系统安全保障; 此外, SGX 可保障运行时的可信执行环境, 恶意代码无法访问与篡改其他程序运行时的保护内容, 进一步增强了系统的安全性; 基于指令集的扩展与独立的认证方式, 使得应用程序可以灵活调用这一安全功能并进行验证. 作为系统安全领域的重大研究进展, Intel SGX 是基于 CPU 的新一代硬件安全机制, 其健壮、可信、灵活的安全功能与硬件扩展的性能保证, 使得这项技术具有广阔的应用空间与发展前景。

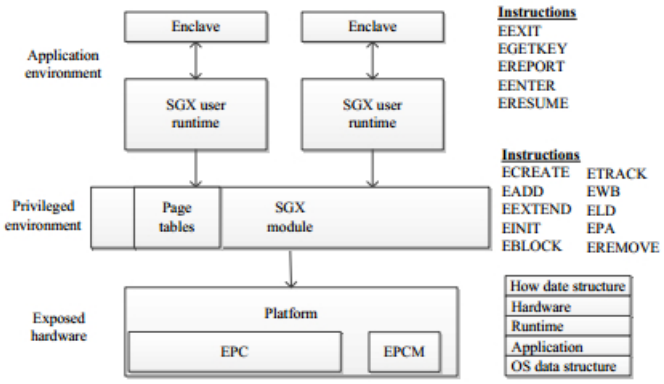

图 1 SGX 架构图

1.2　TCP

TCP 位于传输层，提供可靠的、面向连接的数据传输服务。它确保数据在网络中按顺序、无差错、不丢失、不重复地从源主机传送到目的主机。TCP 通过三次握手建立连接，使用确认应答、超

时重传机制保证可靠性，并通过流量控制和拥塞控制机制动态调整发送速率，防止网络过载或接收方被淹没。

1.3 RC4

RC4（Rivest Cipher 4）是一种经典的流密码加密算法，由 Ron Rivest 于 1987 年设计。它的核心思想是生成一个伪随机密钥流（Keystream），如图2所示，然后将这个密钥流与原始明文（加密时）或密文（解密时）进行逐字节的异或（XOR）运算来得到密文或还原明文。RC4 以其简单、高效和软件实现速度快而闻名，它主要包含两个阶段：密钥调度算法（KSA）用于根据用户提供的可变长度密钥（通常 40-256 位）初始化一个 256 字节的内部状态数组（S 盒）；伪随机生成算法（PRGA）则利用这个初始化后的 S 盒持续生成密钥流字节。

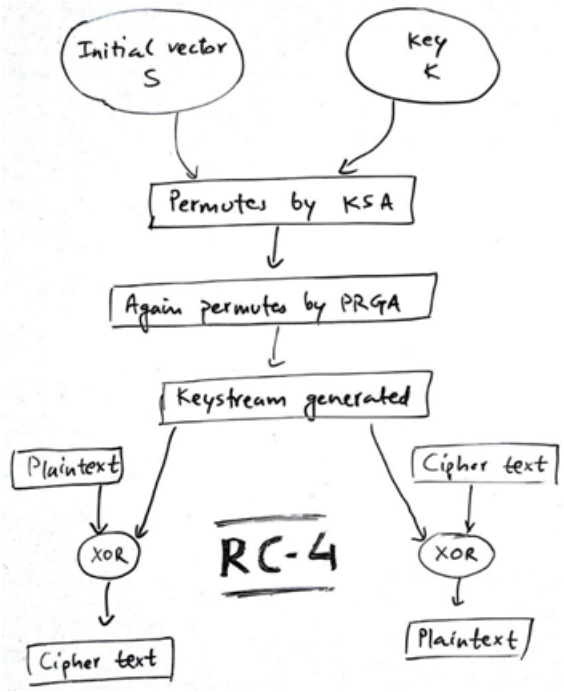

图 2 RC4

2 函数设计

2.1 ocall 函数

ocall 是 enclave 向外部环境请求服务的机制：当 enclave 内的代码需要执行无法或不适合在安全边界内完成的操作时，它会通过发起一个 ocall 来 “回调” 到外部非安全的应用程序或操作系统。ocall 会暂时退出 enclave 的安全保护环境。

在本次实验中需要用到 tcp 发送信息给服务器，从而获得密文，因此这些过程可以在 ocall 中做到，同时标准输出流也可以交给 ocall 来完成。

2.1.1 ocall_connect_to_server

使用 tcp 连接服务器并发送信息需要做到原子化，因此第一步可以先利用 socket 连接到服务器，该函数用来向服务器 (202.120.38.48:41000) 发出连接请求，没有输出以及输入参数。

```
void ocall_connect_to_server()
{
    connect_to_server();
}

int connect_to_server() {
    struct sockaddr_in server_addr;
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("fail to create socket\n");
        return -1;
    }
    // set server address
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    if (inet_pton(AF_INET, SERVER_IP_SJTU, &server_addr.sin_addr)
        <= 0) {
            printf("invalid server IP address");
            close(sock);
            sock = -1;
            return -1;
        }
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(
        server_addr)) < 0) {
        printf("fail to connect to server");
        close(sock);
        sock = -1;
        return -1;
    }
    printf("successfully connect to server\n");
    return 0;
}
```

2.1.2 ocall_get_ciphertext

该函数用来实现向已连接的服务器发送信息，输入是一个用来存储接收密文的 buffer，以及对应的 buffer size，ciphertext_len 是用来表示接收密文的长度。

```
1  void ocall_get_ciphertext(unsigned char *buffer, int buffer_size,
       int *ciphertext_len)
2  {
3      get_ciphertext(buffer,buffer_size, ciphertext_len);
4  }
5
6  int get_ciphertext(unsigned char *buffer, int buffer_size, int *
       ciphertext_len) {
7      if (sock == -1) {
8          printf("Error: Not connected to server\n");
9          return -1;
10     }
11
12     if (buffer == NULL || buffer_size <= 0) {
13         printf("Error: Invalid buffer\n");
14         return -1;
15     }
16
17     memset(buffer, 0, buffer_size);
18     printf("Sending Student ID: %s\n", STUDENT_ID);
19
20     if (send(sock, STUDENT_ID, strlen(STUDENT_ID), 0) < 0) {
21         printf("Error: Failed to send student ID\n");
22         return -1;
23     }
24
25     int bytes_received = recv(sock, buffer, buffer_size, 0);
26     if (bytes_received < 0) {
27         printf("Error: Failed to receive data\n");
28         return -1;
29     } else if (bytes_received == 0) {
30         printf("Error: Server closed connection\n");
31         return -1;
32     }
33
34     *ciphertext_len = bytes_received;
35     return 0;
36 }
```

2.1.3 ocall_disconnect_from_server

该函数用来与服务器断开连接，没有输入输出参数。

```
void ocall_disconnect_from_server(){
    disconnect_from_server();
}

int disconnect_from_server() {
    if (sock != -1) {
        close(sock);
        printf("connection closed\n");
        sock = -1;
    }
    return 0;
}
```

2.1.4 ocall_print_string

该函数用来将 str 打印出来，输入参数是 const char*。

```
void ocall_print_string(const char *str)
{
    /* Proxy/Bridge will check the length and null-terminate
     * the input string to prevent buffer overflow.
     */
    printf("%s", str);
}
```

2.2 ecall 函数

ecall 是进入 enclave 的入口点：当外部应用程序需要 enclave 执行敏感操作或访问其保护的数据时，必须通过发起一个特定的 ecall 来请求服务。这个调用会触发处理器从非安全世界切换到安全世界，执行 enclave 内的受信任代码。

2.2.1 RC4

密钥为 key522031910439。

密钥作为 enclave 的内部变量，不能离开 enclave 内存，解密过程全程在 enclave 中进行，因此会首先在 enclave 中设计密文的解密函数 KSA 和加密函数 PRGA，具体可详见附录。

需要注意的是，从服务器中获取到的密文是 HEX 格式的，因此在具体解密之前需要将十六进制字符串转换为二进制数据，然后再进行 RC4 解密。

2.2.2 ecall_key_enclave

ecall_key_enclave 需要调用 ocall_connect_to_server,ocall_get_ciphertext,ocall_disconnect_from_server,从而与服务器建立连接，获取密文，并断开连接，同时还会用到 ocall_print_string 将明文输出。

```
void ecall_key_enclave() {
    unsigned char ciphertext[MAX_BUFFER_SIZE];
    int ciphertext_len = 0;

    // 1. Connect to server
    ocall_connect_to_server();

    // 2. Get ciphertext (as hex string)
    if (ocall_get_ciphertext(ciphertext, MAX_BUFFER_SIZE, &
        ciphertext_len) != 0) {
        printf("Critical: Failed to get ciphertext\n");
        ocall_disconnect_from_server();
        return;
    }

    printf("Received ciphertext (%d bytes)\n", ciphertext_len);

    // 输出原始接收到的十六进制密文
    printf("Hex Ciphertext: %s\n", ciphertext);

    // 3. Convert hex string to binary data
    unsigned char binary_ciphertext[MAX_BUFFER_SIZE/2]; // 十六进
        制字符串转二进制后长度减半
    int binary_len = hex_string_to_binary((const char *)
        ciphertext, binary_ciphertext, sizeof(binary_ciphertext));

    if (binary_len <= 0) {
        printf("Error: Invalid hex string or buffer too small\n")
            ;
        ocall_disconnect_from_server();
        return;
    }

    printf("Converted to binary (%d bytes)\n", binary_len);

    // 4. Prepare decryption
```

```
33      const char *key = STUDENT_ID_key;
34      int key_len = strlen(key);
35
36      // 5. Perform RC4 decryption
37      rc4_init((const unsigned char *)key, key_len);
38
39      unsigned char *temp_buffer = (unsigned char *)malloc(
            binary_len + 1); // +1 for null terminator
40      if (!temp_buffer) {
41          printf("Critical: Memory allocation failed\n");
42          ocall_disconnect_from_server();
43          return;
44      }
45
46      memcpy(temp_buffer, binary_ciphertext, binary_len);
47
48      rc4_crypt(temp_buffer, binary_len);
49
50      // Null-terminate the plaintext
51      temp_buffer[binary_len] = '\0';
52
53      // 6. Output decrypted result via Ocall
54      printf("Decrypted Result: %s\n", temp_buffer);
55
56      // 7. Cleanup
57      free(temp_buffer);
58      ocall_disconnect_from_server();
59  }
```

3 实验结果

如图3所示，接收到的密文为 64fe6b9abb0f078e008b139fd857c2825a4da51a92610c8849a33e70，密钥为 key522031910439，经过解密后成功获得明文：Intel_SGX_Flag{420951902133}

同时为了确定结果是否准确，依照 GoSec 官网要求修改学号为 862154749110，得到了明文 Intel_SGX_Flag{942181047561}，如图4所示，结果正确：

```
whitefork@whitefork-VirtualBox:~/Desktop/sgx$ ./app
successfully connect to server
Sending Student ID: 522031910439
Received ciphertext (56 bytes)
Hex Ciphertext: 64fe6b9abb0f078e008b139fd857c2825a4da51a92610c8849a33e70
Converted to binary (28 bytes)
Decrypted Result: Intel_SGX_Flag{420951902133}
connection closed
Info: VerySampleEnclave successfully returned.
```

图 3 Result

图 4 Result on GoSec

A 附录

B RC4 设计与解密加密函数

```c
// RC4 Implementation
struct rc4_state {
    unsigned char S[256];
    int i, j;
};

static struct rc4_state state;

// KSA
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
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;
}
```

参考文献

[1] 程越强赵波韦韬严飞张焕国 马婧王鹃, 樊成阳. Sgx 技术的分析和研究. 软件学报, 29(9):2778, 2018.