A Collection of Senselock EL Case-Study Samples

Feb 2012

Version 1.0.4

SENSELOCK
Reliable Professional Partner

1. PREFACE **1**

1.1. DESCRIPTION OF SAMPLE STRUCTURE 1
1.2. INTRODUCING COMMON FUNCTIONS 2
1. OPENS4BYINDEX 2
2. RESETANDCLOSES4 3
3. HEXPRINT 3

2. BASIC OPERATIONS **4**

SAMPLE 1: WHEN THERE ARE SEVERAL SENSELOCK EL DEVICES, HOW TO IDENTIFY AND MANAGE YOUR OWN DEVICES USING "CUSTOMER ID"? 5
Sample Gist *5*
Sample Introduction *5*
Sample Implementation *5*
SAMPLE 2: HOW TO IDENTIFY AND MANAGE YOUR OWN SENSELOCK EL DEVICES USING DONGLE ID? *7*
Sample Gist *7*
Sample Introduction *7*
Sample Implementation *7*
Questions *8*
SAMPLE 3: HOW TO RESET DEVICE? 9
Sample Gist *9*
Sample Introduction *9*
Sample Implementation *9*
Questions *9*
SAMPLE 4: HOW TO USE HID MODE (DRIVER INSTALLATION NOT REQUIRED)? 9
Sample Gist *10*
Sample Introduction *10*
Sample Implementation *11*
Questions *11*
SAMPLE 5: HOW TO CONTROL LED INDICATOR OF THE SENSELOCK EL DEVICE? *11*
Sample Gist *12*
Sample Introduction *12*
Sample Implementation *12*
Questions *12*
SAMPLE 6: HOW TO GET STORAGE SPACE OF THE DEVICE AND REMNANT SPACE OF CURRENT DIRECTORY? *12*
Sample Gist *13*
Sample Introduction *14*
Sample Implementation *14*
Questions *14*
SAMPLE 7: HOW TO GET DEVICE TYPE? *14*
Sample Gist *15*
Sample Introduction *15*
Sample Implementation *15*
Questions *15*
SAMPLE 8: HOW TO SET DEVELOPER PIN AND USER PIN USING API? *15*
Sample Gist *16*
Sample Introduction *16*
Sample Implementation *16*
Questions *16*
SAMPLE 9: HOW TO USE API TO OPERATE DIRECTORY? *16*
Sample Gist *17*
Sample Introduction *17*
Sample Implementation *17*
Questions *17*
SAMPLE 10: HOW TO OPERATE FILES USING API? 17
Sample Gist *18*
Sample Introduction *18*
Sample Implementation *18*
SAMPLE 11: PROTECT MULTIPLE SOFTWARE WITH MULTIPLE DIRECTORIES *19*

Sample Gist.....20
Sample Introduction.....20
Sample Implementation.....20
Questions.....22
SAMPLE 12: HOW TO INSTALL/UNINSTALL SENSELOCK EL DRIVER IN YOUR OWN CODES?.....22
Sample Gist.....23
Sample Introduction.....23
Sample Implementation.....23
Questions.....24
SAMPLE 13: HOW TO REALIZE DONGLE INSERT/REMOVE NOTIFICATION?.....24
Sample Gist.....25
Sample Introduction.....25
Sample Implementation.....25
Questions.....27
SAMPLE 14: HOW TO OPERATE DEVICE IN MULTITHREAD?.....27
Sample Gist.....28
Sample Introduction.....28
Sample Implementation.....28
Questions.....28
SAMPLE 15: HOW TO OPERATE DEVICE IN MULTIPROCESS?.....29
Sample Gist.....30
Sample Introduction.....30
Sample Implementation.....30
Questions.....30

3. ABOUT CODE TRANSPLANT.....31

SAMPLE 16: HOW TO ACCOMPLISH INPUT/OUTPUT USING COMMUNICATION BUFFER?.....32
Sample Gist.....33
Sample Introduction.....33
Sample Implementation.....34
Questions.....40
SAMPLE 17: HOW TO USE CODE MEMORY?.....40
Sample Gist.....41
Sample Introduction.....41
Sample Implementation.....41
Questions.....42
SAMPLE 18: HOW TO USE SHARED MEMORY?.....42
Sample Gist.....43
Sample Introduction.....43
Sample Implementation.....44
Questions.....45
SAMPLE 19: HOW TO USE HARDWARE FUNCTION LIBRARY?.....46
Sample Gist.....47
Sample Introduction.....47
Sample Implementation.....47
Questions.....47

4. FILE OPERATIONS.....48

SAMPLE 20: HOW TO READ & WRITE A SENSELOCK EL DATA FILE?.....49
Sample Gist.....50
Sample Introduction.....50
Sample Implementation.....50
Questions.....52
SAMPLE 21: HOW TO CREATE A FILE DYNAMICALLY IN THE DONGLE?.....53
Sample Gist.....54
Sample Introduction.....54
Sample Implementation.....55
Questions.....56

5. MATHEMATICAL OPERATIONS.....57

SAMPLE 22: HOW TO USE SINGLE PRECISION FLOATING POINT NUMBER?.....58
Sample Gist.....59
Sample Introduction.....59
Sample Implementation.....59
Questions.....60
SAMPLE 23: HOW TO USE DOUBLE PRECISION FLOATING POINT NUMBER?.....60
Sample Gist.....61
Sample Introduction.....61
Sample Implementation.....61
Questions.....63
SAMPLE 24: HOW TO AVOID DOUBLE PRECISION FLOATING POINT NUMBER BOUNDARY INACCURACY?63
Sample Gist.....64
Sample Introduction.....64
Sample Implementation.....64
Questions.....66

6. CRYPTOGRAPHIC ALGORITHMS.....66

SAMPLE 25: HOW TO ENCRYPT DATA USING DES/TDES?.....67
Sample Gist.....68
Sample Introduction.....68
Sample Implementation.....69
Questions.....73
SAMPLE 26: HOW TO USE RSA FOR DATA ENCRYPTION/DECRYPTION, DIGITAL SIGNATURE AND VERIFICATION?.....73
Sample Gist.....74
Sample Introduction.....74
Sample Implementation.....75
Questions.....78
SAMPLE 27: HOW TO WORK OUT H-MAC MESSAGE AUTHENTICATION CODE?.....78
Sample Gist.....79
Sample Introduction.....79
Sample Implementation.....79
Questions.....80
SAMPLE 28: HOW TO WORK OUT DES-MAC CODE?.....80
Sample Gist.....81
Sample Introduction.....81
Sample Implementation.....81
Questions.....82
SAMPLE 29: HOW TO IMPLEMENT MD5 ALGORITHM IN SENSELOCK EL?.....82
Sample Gist.....83
Sample Introduction.....83
Sample Implementation.....84
Questions.....85
SAMPLE 30: HOW TO USE RANDOM NUMBERS IN SIMPLE COMMUNICATION ENCRYPTION?.....85
Sample Gist.....86
Sample Introduction.....86
Sample Implementation.....87
Questions.....88
SAMPLE 31: HOW TO USE ENCRYPTED DATA FILES?.....88
Sample Gist.....89
Sample Introduction.....89
Sample Implementation.....89
Questions.....91

7. TIPS (ANSWERS) TO QUESTIONS.....91

SAMPLE 2.....92
SAMPLE 4.....93
SAMPLE 6.....93
SAMPLE 8.....93
SAMPLE 11.....94

SAMPLE 14 94
SAMPLE 15 94
SAMPLE 16 94
SAMPLE 17 95
SAMPLE 18 95
SAMPLE 19 95
SAMPLE 20 95
SAMPLE 21 96
SAMPLE 22 96
SAMPLE 25 97
SAMPLE 26 97
SAMPLE 27 97
SAMPLE 28 98
SAMPLE 30 98
SAMPLE 31 98

1. Preface

As a supplementary course for *Senselock EL Developer Guide*, this document includes a number of practical samples of Senselock EL. Readers, according to their own situations, may select samples relevant to their product for reference to better software protection. This document contains some general and universal samples that are supposed to be helpful for all of the software developers using Senselock EL.

Please bear in mind that the samples are not designed to be used directly by readers, but, hopefully, to help and enlighten users. To understand the purpose of these samples, modify them and then use them are the appropriate ways we recommend.

1.1. Description of Sample Structure

To simplify reading and search, this manual makes a unified structure of all the samples, including description format and source code illustration.

Each sample contains the following parts:

- Sample gist — Specifies the main purpose of the sample, and indicates the goal that can be achieved through this sample. Readers shall read this part carefully to make sure whether this sample is useful to them.
- Sample introduction — Introduces the sample or describes the functions of the sample.
- Sample implementation — Illustration for code implementation, which, generally speaking, involves two parts: Senselock EL code and PC code. The former specifies the internal code of Senselock EL hardware to be composed while the latter is the corresponding code in PC end. This part usually only contains the most important segments of sample source codes. The full package of source codes is available on CD of SDK.
- Questions — One or several questions are designed to check how well readers understand samples.

The root directory of the sample contains the following subdirectories:

- *common* — common function code
- *inc* — header file used by almost all PC ends and Senselock EL hardware codes
- *lib* — static library used by almost all PC ends and Senselock EL hardware codes

The source codes of all samples are available in the CD directory of *<SDK>\samples\case study*. Each sample is stored in an independent directory, which usually breaks into three subdirectories:

- *inc* — common header file, namely, the one used by Senselock EL hardware codes and PC codes.
- *win32* — PC source codes, usually offered by standard C version.
- *hardware* — internal source codes of Senselock EL hardware.

1.2. Introducing Common Functions

Some functions, such as, "open Senselock EL device", are used in almost all Senselock EL programs. For the sake of convenience, some common functions are encapsulated into independent ones. There are currently three of them, defined in *"<SDK>\samples\case study\common\common.c"*.

`OpenS4ByIndex()`	Use "share" mode to open the Senselock EL device of specified index and return the corresponding device context.
`ResetAndCloseS4()`	Reset and close Senselock EL device.
`hexprint()`	Print the byte array of a certain length by hexadecimal number.

Given below is a description of them one by one.

1. OpenS4ByIndex

Use "share" mode to open the Senselock EL device with specified index and return the corresponding device context.

Parameter:

index	[in] target device index.
pS4CtxList	[out] pointer to SENSE4_CONTEXT structure.

Returned value:

If the function is successfully invoked, return 1 (TRUE), otherwise return 0 (FALSE).

Remarks:

For the convenience of program writing, the initial operation sequence (list, open device) that must be invoked by any Senselock EL program is encapsulated.

Implementation Process:

1. Enumerate/search Senselock EL devices.
   It is usually necessary to invoke `S4Enum()` twice. At the first time, set the first parameter of `S4Enum()` as NULL so that the second parameter will return the storage space actually needed to the variable specified by *pdwSize*. After the first call, it can obtain the number of the Senselock EL devices currently connected to the HOST and the size of device context list (The number of devices can be worked out in this way: `Num = *pdwSize/sizeof (SENSE4_CONTEXT))`. Allocate dynamically the required memory to save the context list of the devices (or allocate statically a sufficiently large memory space) and invoke again the `S4Enum()` function to copy the contexts of all the devices to the list. Finally, copy the device context of Senselock EL with specified *index* to the returned parameter. Free dynamically allocated memory.

2. Open specified device.
   Simply pass the context pointer of the specified device to `S4Open()`

function as a parameter to open this device.

2. ResetAndCloseS4

Reset and close Senselock EL device.

Parameter:

pS4CtxList [in] Pointer to SENSE4_CONTEXT structure.

Returned value:

None.

Remarks:

After `S4VerifyPin()` has been successfully called, even S4Close() is also called successfully, the security status inside the dongle will not change. For better security when closing a device, it's recommended to use this function instead of merely calling `S4Close()`.

Implementation process:

1. Invoke `S4Control (…S4_RESET_DEVICE…);`
2. Invoke `S4Close ();`

3. hexprint

Print the byte array of a certain length by hexadecimal number.

Parameter:

Fp [in] file pointer, usually set as `stdout`.
Data [in] array pointer.
Len [in] array length.

Returned value:

None.

2. Basic Operations

This chapter introduces the basic operations of Senselock EL equipment, including the most basic access modes, operation procedure, information acquisition as well as some tips and skills. This part does not involve code transplant.

Sample 1: When there are several Senselock EL devices, how to identify and manage your own devices using "Customer ID"?

Sample Gist

1. The importance of identifying and managing your own Senselock EL devices.

   If there are several Senselock EL devices on the local system, it is imperative to open the correct device because of two reasons: first, connecting to the wrong device will make the software misinterpret the dongle as illegal and thus lead to software running error; second, operating the dongle of aother software might modify the security status of that dongle, thus impacting the normal operation of that software. To avoid these problems and open the correct dongle, one of the two common approaches can be employed:

   Approach 1: use dongle's "Customer ID" (also known as "Developer ID").

   Senselock EL can offer the service of setting "Customer ID". We can provide different customers with different 4-byte customer IDs, which we will ensure its uniqueness and its value can be obtained via `S4Control()` function. The software thus can distinguish different devices according to their customer IDs. This sample will introduce briefly the usage of customer ID. For the details and description of the `S4Control()` function, please refer to Chapter 9 in *Senselock EL Developer Guide*.

   Approach 2: Create and manage your own dongle ID.

   This is the most feasible approach. While creating the root directory for dongle, you can create a user-defined ATR file, the content of which is a user-defined 8-byte ID. When the `S4Enum()` function is run, the IDs of all dongles will be stored in the *bID* member variable of `SENSE4_CONTEXT` structure. In this case, devices can be differentiated from each other according to their IDs. For the details of ATR file creation, please refer to the description of `S4CreateDirEx()` function. For the actual implementation of this approach, please see sample2.

2. What is the difference between customer ID and dongle ID?

   The customer ID is offered in additional service given by Senselock to customers with this requirement. It will be written in as a read-only and unique 4-byte number in the manufacture of the dongle. The dongle ID is any 8-byte data set by the developer himself. Although there is no way to guarantee its uniqueness in theory, the chance of repeating itself in practice is very low.

Sample Introduction

This sample demonstrates how to enumerate a number of Senselock EL devices and get customer ID of each device. Notice that the customer ID is 4-byte all-zero data by default. Only after the software developer receives "setting customer ID" service from Senselock will it be of practical meaning to get this value.

Sample Implementation

PC End Program

The program will enumerate one by one all the Senselock EL devices inserted to the PC, get the customer ID of each device and print them out one by one. The process of enumerating devices is similar to the `OpenS4ByIndex()` function. No detailed description will be given here. Given below is the code for getting customer ID:

```
dwResult = S4Control(pstS4CtxList+I, S4_GET_CUSTOMER_NAME, NULL, 0,
                     lpOutBuf, sizeof(lpOutBuf), &dwBytesReturned);

// check dwResult here
```

If the inserted Senselock EL dongle is not set with customer ID, the above-mentioned code

will return 4-byte “0x00” in the lpOutBuf.

Sample 2: How to identify and manage your own Senselock EL devices using dongle ID?

Sample Gist

1. The importance of identifying and managing your own Senselock EL devices.

   Please see the description in the Sample Gist of sample1.

2. What is Senselock EL dongle ID? how to self-define dongle ID?

After the specific Senselock EL device is listed by calling `S4Enum()`, the member variable *bID* of the device's context structure contains an 8-byte ID called "dongle ID". By default, the ID is as same as the unique serial number of Senselock EL hardware. Unique serial number does not possess regularity and software developer may self define the dongle ID to make the management of software easy. The following two methods can be used to set the dongle ID:

- While creating the root directory for the dongle, you can create a self-defined ATR file, which contains a user-defined 8-byte dongle ID. For the details of the ATR file, please see the description of `S4CreateDirEx()` function in *Senselock EL Developer Guide*.
- Invoke Senselock EL API `S4Control (…S4_SET_DEVICE_ID…)` to set dongle ID directly.

3. Some Precautions

- When the root directory of the dongle is deleted, the ATR file will be deleted simultaneously, so the dongle ID will be recovered to "unique serial number" of the hardware.
- After the dongle ID is set, it is necessary to re-plug or reset the dongle so as to update the data of member variable *bID* of the device's context structure.
- The developer level authorization is required in the setting of dongle ID.

Sample Introduction

This sample contains two programs. Run first *sample_2_set.exe*, which demonstrates how to set one or multiple dongle IDs, then run *sample_2_get.exe*, which prints all the connected Senselock EL dongle IDs one by one to demonstrate the setting effect.

In the sample, we set the dongle ID as an 8-byte character string: "OurS4_**", where ** indicates hexadecimal serial number. For example, the ID of the first dongle is "OurS4_00", the ID of No.11 dongle is "OurS4_0b", etc.

Sample Implementation

PC End Program

First check the implementation code of setup program *sample_2_set.exe*, which will list one by one all the Senselock EL devices inserted into the PC and define dongle ID for each device.

If there is no root directory, `S4CreateEx()` will be invoked to generate the root directory as well as creating dongle ID.

```
...
else if (dwResult == S4_DEVICE_UNSUPPORTED)// no root dir
{
     S4CREATEDIRINFO s4_CreateInfo;
     s4_CreateInfo.dwS4CreateDirInfoSize = sizeof(S4CREATEDIRINFO);
```

```
    memcpy(s4_CreateInfo.szAtr, lpCustomerID, 8);

    dwResult = S4CreateDirEx(pstS4CtxList+i, "\\", 0,
                             S4_CREATE_ROOT_DIR,
                             &s4_CreateInfo);
    ...// check dwResult here
}
```

If the root directory does exist, the following code will be invoked to recreate dongle ID.

```
dwResult = S4VerifyPin(pstS4CtxList+i, lpDevPin, 24, S4_DEV_PIN);
...// check dwResult here

dwResult = S4Control(pstS4CtxList+i, S4_SET_DEVICE_ID, lpCustomerID, 8,
                     NULL, 0, NULL);
...// check dwResult here
```

Then check the implementation code of *sample_2_get.exe* that gets the dongle ID. The program will list one by one all the Senselock EL devices inserted into PC. Notice that it is necessary to re-plug the hardware or reset the device before the updated dongle ID information is visible. The code below resets the hardware dongle, reads and prints the current dongle ID:

```
/* need reset to get the new bID */
dwResult = S4Control(pstS4CtxList+i, S4_RESET_DEVICE, NULL, 0,
                     NULL, 0, NULL);
...// check dwResult here

memcpy(buffer, &((pstS4CtxList+i)->bID), 8);
memcpy(buffer+8, "\0", 1);
printf("The Customer ID for device No.%d is:%s\n", i, buffer);
```

As for how to reset the device, please see sample3.

Questions

Q1. What are the difference and relationship between self-defined dongle ID, customer ID and unique serial number of the hardware?

Q2. If the self-defined dongle ID is set, is the unique serial number of the hardware still useful? How to get the unique serial number of the hardware?

Sample 3: How to reset device?

Sample Gist

1. Why do we reset device? How to reset device?

   Resetting device fulfills the purposes as follows:

   - Simply invoking `S4Close()` will not clear the security status. Before closing Senselock EL, resetting device can clear the security status in the dongle, thus removing some hidden security troubles.
   - Refresh the information of some devices, such as the newly set "dongle ID" information. For detailed instance, please see sample2.

   Resetting device requires the invocation of the API function: `S4Control (…S4_RESET_DEVICE…)`

Sample Introduction

This sample demonstrates briefly how to reset Senselock EL device.

Sample Implementation

PC End Program

The code below accomplishes the function of resetting device:

```
printf("Reset Senselock EL device  ...\n");

dwResult = S4Control(&stS4Ctx, S4_RESET_DEVICE, NULL, 0, NULL, 0, NULL);
...// check dwResult here
```

Questions

Sample 4: How to use HID mode (driver installation not required)?

Sample Gist

1. What is HID mode?

HID is short for Human Interface Device. Senselock EL 2.3 can run like a HID device. Since the Windows operating system is already installed with the standard driver of the HID device, Senselock EL can run without the need for installing extra driver.

Senselock EL2.3 supports HID mode as well as standard USB mode and can switch between two modes freely without impacting the running of the software by just sending the control code command. Notice that the device shall be plugged once again after switchover before the setting takes effect.

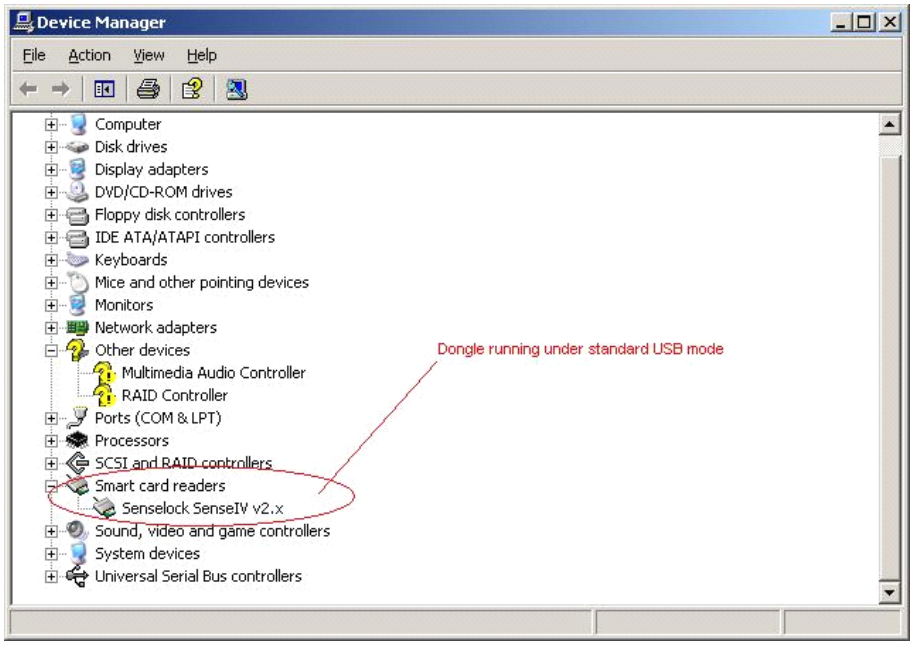
2. What are the advantage and disadvantage of HID mode and USB mode?

The advantage of the HID mode is that in the operating system above Windows98, there is no need to install the driver, but it is slightly inferior to the standard USB mode in capability and efficiency (for example, it does not support multi-process access) and is supported only by Senselock EL 2.3 or higher, so please exercise caution when using it.

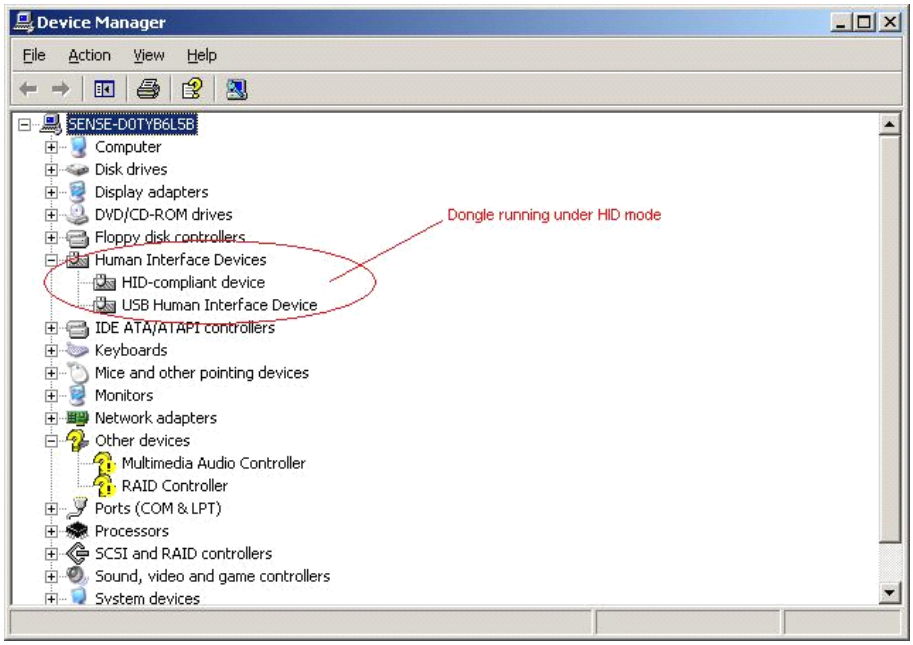
Sample Introduction

This sample demonstrates how to set the Senselock EL device so that it can run under different drive modes: HID mode or standard USB mode.

After the program is run, choose to input "1", Senselock EL will be set to run under standard USB mode. When the setting is complete, re-plug the dongle and Senselock EL device in the "Device Manager" will show up under the node of "Smart Card Reader":



Choose to input "2" and Senselock EL will be set to run under the HID mode. After the setting is complete, re-plug the device and Senselock EL device in the "Device Manager" will show up under the node of HID:



Sample Implementation

PC End Program

The code below makes Senselock EL run under the USB mode:

```
dwResult = S4Control(&stS4Ctx, S4_SET_USB_MODE, NULL, 0, NULL,
                         0, NULL);
```

The code below makes Senselock EL run under the HID mode:

```
dwResult = S4Control(&stS4Ctx, S4_SET_HID_MODE, NULL, 0, NULL,
                           0, NULL);
```

Questions

Q1. If devices in HID mode and in USB mode are plugged in the PC concurrently, can they operate normally at the same time?

Sample 5: How to control LED indicator of the Senselock EL device?

Sample Gist

1. What are the purposes of LED indicator of the Senselock EL device? How to control it?

The Senselock EL device contains a LED, which can be controlled by software and designed to display some status information. By default, whenever a Senselock EL device is inserted, the LED indicator will be ON, which indicates it can operate normally. The software can set LED to ON/OFF or wink at specified frequency.

Sample Introduction

This sample demonstrates how to write codes to control the LED behaviors of Senselock EL. First, the LED will wink 5 times/sec. After the user press any key, it will be OFF. After the user presses any key again, it will be ON.

Sample Implementation

PC End Program

The code below makes Senselock EL "wink" at the specified frequency:

```
bWinkFreq = 5;

dwResult = S4Control(&stS4Ctx, S4_LED_WINK, &bWinkFreq,
                         1, NULL, 0, NULL);
```

The code below makes Senselock EL "OFF":

```
dwResult = S4Control(&stS4Ctx, S4_LED_DOWN, NULL, 0, NULL, 0, NULL);
```

The code below makes Senselock EL "ON":

```
dwResult = S4Control(&stS4Ctx, S4_LED_UP, NULL, 0, NULL, 0, NULL);
```

Questions

Sample 6: How to get storage space of the device and remnant space of current directory?

Sample Gist

1. About storage space of the device and remnant space of current directory.

The device storage space is the storage size of the whole Senselock EL device. Senselock EL2.3 can offer storage spaces of different specifications: 8K, 32K and 64K. You can get the storage space of the current device using Senselock EL tools or get this information using Senselock EL API.

The remnant space of the directory means how much space is available in the current directory. The Senselock EL device needs to specify the size of the directory before creating it and the value can not be changed after it is created. While creating the directory, the system file will seize some space. Sometimes, we need to know how much space is available in the current directory.

The information can be obtained by invoking the `S4Control()` function to send control code.

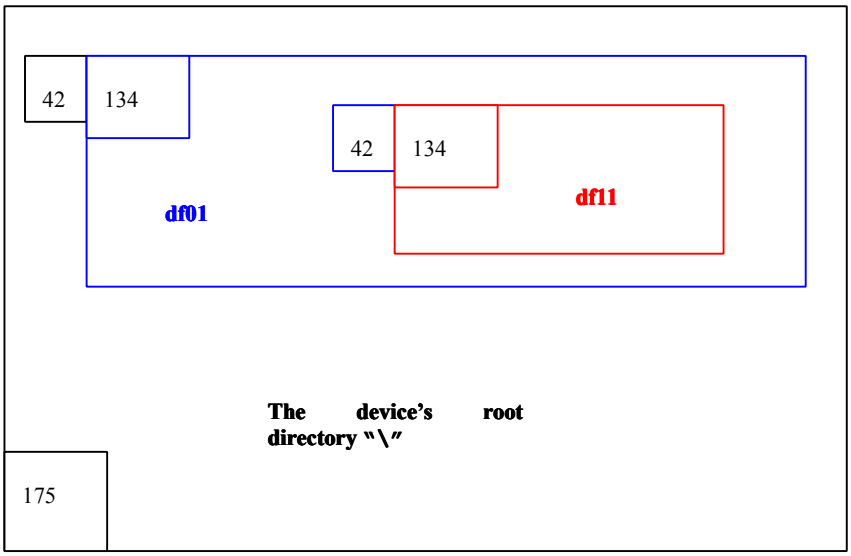
2. About the device storage structure.

The diagram below might help you better understand the storage structure of Senselock EL desktop version. The diagram shows a dongle of 64K model whose storage space is 64575 (0xfc3f) bytes. After the root directory is created, the system files seizes 175 bytes. Therefore, the remnant space of the root directory is 64575 – 175 = 64400 (0xfb90). Then, create a 20000-byte directory “df01”. The system file again will seize 134 bytes and meanwhile the parent directory of the new directory will be seized 42 bytes for the purpose of management. In this case,

Remnant space of the root directory	64400 – 20000 – 42 = 44358（0xad46）
Remnant space of df01 directory	20000 – 134 = 19866

After that, create a 10000-byte sub-directory “df11” under “df01”. In this case:

Remnant space of the root directory	44358 (unchanged)
Remnant space of df01 directory	19866 – 10000 – 42 = 9824
Remnant space of df11 directory	10000 – 134 = 9866



For the detailed introduction to directory creation, please see the description of `S4CreateDir()` function in Chapter 9 of *Senselock EL Developer Guide*.

Sample Introduction

This sample demonstrates how to get the storage space of the device and the remnant space of the current directory by invoking Senselock EL API.

Sample Implementation

PC End Program

The code below gets the storage space of the device:

```
dwResult = S4Control(&stS4Ctx, S4_GET_DEVICE_USABLE_SPACE,
            NULL, 0,  &wSpaceSize, sizeof(wSpaceSize), NULL);
```

The code below gets the remnant space of the current directory:

```
dwResult = S4Control(&stS4Ctx, S4_DF_AVAILABLE_SPACE, NULL,
    0, &wSpaceSize, sizeof(wSpaceSize), NULL);
```

Questions

Q1. In the example as mentioned above, to create a subdirectory "df21" under "df11", can we set the size of new directory as the "remnant space" (9866 bytes) of "df11"?

Sample 7: How to get device type?

Sample Gist

1. How many device types does Senselock EL have? How to differentiate them?

There are three device types of Senselock EL: desktop version dongle, network master dongle and network slave dongle. The type of the current device can be obtained by invoking Senselock EL API.

Sample Introduction

This sample demonstrates how to get the device type of a dongle.

Sample Implementation

PC End Program

The code below gets the device type and prints it out:

```
dwResult = S4Control(&stS4Ctx, S4_GET_DEVICE_TYPE, NULL,
            0, &bDeviceType, sizeof(bDeviceType), NULL);
...//Check dwReslt

switch(bDeviceType)
{
    case S4_LOCAL_DEVICE:
        printf("Desktop version.\n");
    break;

    case S4_MASTER_DEVICE:
        printf("Network version - master device.\n");
    break;

    case S4_SLAVE_DEVICE:
        printf("Network version - slave device.\n");
    break;

    default:
        printf("Unexpected device type!\n");
```

Questions

Sample 8: How to set developer PIN and user PIN using API?

Sample Gist

1. The importance of modifying developer PIN!!!

   Despite the fact that the internal data of Senselock EL hardware cannot be read directly with the developer PIN, the leakage of the developer PIN will still threaten critically the security of the data in Senselock EL. When Senselock EL is used for the first time, its default developer PIN is "123456781234567812345678". Before Senselock EL is released to final users, make sure to modify it to a secret value.

   Make sure to modify the the default developer PIN under such circumstances:

   - Re-initialize the file system.
   - Create a directory.
   - Clear all the files and subdirectories under a directory. In this case, the developer PIN for this directory will be reset to default value.

2. Please note that under the following situations, the security status in the dongle is at "developer" level. At this point, the gate of the dongle is open! Please make sure to use the reset command in a timely manner or re-plug the dongle to clear this status.

   - The developer PIN is just modified.
   - The developer PIN is verified with `S4VerifyPin()`.
   - The root directory is newly created with `S4CreateDirEx()`.

   For the introduction to the Senselock EL security system, please see Chapter 1 of *Senselock EL Developer Guide*.

Sample Introduction

This sample demonstrates how to modify developer PIN and user PIN by invoking Senselock EL API.

Sample Implementation

PC End Program

The code below demonstrates how to modify developer PIN and user PIN:

```
dwResult = S4ChangePin(&stS4Ctx, lpOldPin, dwOldPinLen,
                lpNewPin, dwNewPinLen, S4_DEV_PIN);
...// check dwResult

dwResult = S4ChangePin(&stS4Ctx, lpOldUsrPin, dwOldUsrPinLen,
                lpNewUsrPin, dwNewUsrPinLen, S4_USER_PIN);
...// check dwResult
```

Questions

Q1. What role does the user PIN play? Why not emphasize its security?

Q2. Why is it not recommended to differentiate your own devices using user PINs in the event of multiple Senselock EL dongles?

Sample 9: How to use API to operate directory?

Sample Gist

1. What basic operations are there in terms of directory?

   Developer level operations like creating and deleting directory and user level operations like switching between directories.

2. Some precautions

   - Never ever use the "developer level" API function in the software sent to end users!!
   - After creating or clearing directory, do not forget to modify the developer PIN!!!
   - While creating a directory, notice that some of the IDs are retained by the system and cann't be used any more. These retained IDs include 0x0000, 0x0015, 0x0016, 0x0018, 0x001e, 0x3f00, 0x3f01, 0x3f02, 0x3f03 and 0x3f04.

Sample Introduction

This sample demonstrates how to operate the directory of Senselock EL. First, try to enter the target directory. If it exists, clear it. If not, create this directory and then modify the developer PIN and user PIN under this directory.

Sample Implementation

PC End Program

The code below tries to switch to the target directory:

```
dwResult = S4ChangeDir(&stS4Ctx, DEST_DIR);
```

If it exists, clear it:

```
/* verify DEV_PIN first */
dwResult = S4VerifyPin(&stS4Ctx, lpNewPin, dwNewPinLen,
           S4_DEV_PIN);
...// check dwResult

dwResult = S4EraseDir(&stS4Ctx, NULL);
...// check dwResult
```

If not, create the directory. When the directory is created, for the relations between directory size and device storage space, please see sample 6.

```
/* verify DEV_PIN first */
dwResult = S4VerifyPin(&stS4Ctx, lpNewPin, dwNewPinLen,
               S4_DEV_PIN);
...// check dwResult

dwResult = S4CreateDir(&stS4Ctx, DEST_DIR, dwSubDirSize,
               S4_CREATE_SUB_DIR);
...// check dwResult
```

Do remember to modify the default developer PIN under the directory after the above operation is completed. For details on PIN modification, please see sample 8.

Questions

Sample 10: How to operate files using API?

Sample Gist

1. What basic operations are there for files?

Developer level operations like creating/modifying file and user level operations like invoking executable files or getting file information.

2. Some precautions

- Never ever use "developer level" API function in the software sent to end users!!
- It is better to add the option "`S4_FILE_EXECUTE_ONLY`" while creating Senselock EL executable files.
- Once a file is created, it cannot be deleted directly. To wipe out the file and free the storage space it seizes, the directory where the file is located shall be cleared/deleted.
- While creating a directory, notice that some of the IDs are retained by the system and cann't be used any more. These retained IDs include 0x0000, 0x0015, 0x0016, 0x0018, 0x001e, 0x3f00, 0x3f01, 0x3f02, 0x3f03 and 0x3f04.

Sample Introduction

This sample demonstrates how to operate Senselock EL files using API. First, write a hardware program file (Senselock EL EXF) on the disk to Senselock EL, then get the information of the newly written hardware program file and finally process some input data by invoking it.

Sample Implementation

Compile and link the Senselock EL Executable file (EXF) program as mentioned below. Copy the successfully linked hex file *sample_10_hw.hex* to the directory of *..\sample_10\win32* and then debug and run the PC end program (To run the PC end program directly, *sample_10_hw.hex* shall be placed in the same directory as the program). As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Senselock EL EXF Program

For the sake of convenience, the hardware program uses the "bubble sort" program as mentioned in Chapter 3 of *Senselock EL Developer Guide*:

```
#include "ses_v3.h"

/*bubble sort function*/
void bubble_sort(unsigned char *p, int len)
{
    int i,j;
    unsigned char tmp;

    for (i=0;i<len-1;i++)
    {
        for (j=0;j<len-i-1;j++)
        {
            if (p[j] < p[j+1])
            {
                tmp = p[j];
                p[j] = p[j+1];
                p[j+1] = tmp;
            }
```

```
        }
    }
}

/*Senselock EL main procedure*/
void main()
{
    unsigned char *test = pbInBuff;
    int len = bInLen;

    bubble_sort(test, len);

    _set_response(len,test);
    _exit();
}
```

PC End Program

The code below writes the hardware program directly from PC hard disk to Senselock EL and meanwhile sets its attribute as "internal non-readable/non-writable" (with flag `S4_FILE_EXECUTE_ONLY`):

```
/* verify DEV_PIN first */
dwResult = S4VerifyPin(&stS4Ctx, lpNewPin, dwNewPinLen,
                S4_DEV_PIN);
...// check dwResult

dwResult = PS4WriteFile(&stS4Ctx, S4_EXE_FILE_ID,
                "sample_10_hw.hex", &dwFileSize,
                S4_CREATE_NEW| S4_FILE_EXECUTE_ONLY,
                S4_HEX_FILE, &dwBytesWritten);
...// check dwResult
```

After that, invoke the following code to get the information of the newly written Senselock EL files:

```
dwResult = S4Control(&stS4Ctx, S4_EF_INFO, S4_EXE_FILE_ID, 4,
                &stEFInfo, sizeof(EFINFO), &dwBytesReturned);
```

Finally, invoke the newly written executable file, process input data: make sorting operation:

```
/* verify USER_PIN first */
dwResult = S4VerifyPin(&stS4Ctx, lpUsrPin, dwUsrPinLen,
                        S4_USER_PIN);
...// check dwResult

dwResult = S4Execute(&stS4Ctx, S4_EXE_FILE_ID, test,
        sizeof(test), test, sizeof(test), &dwBytesReturned);
...// check dwResult
```

Questions

Sample 11: Protect Multiple Software with Multiple Directories

Sample Gist

1. Why do we use multiple directories?

All PINs, be it developer PIN or user PIN, are designed for an internal directory of Senselock EL, but not for the whole Senselock EL device. Each directory contains its own developer PIN and user PIN. The security attributes between directories are independent. This feature allows several irrelevant software to be stored under multiple different directories, protected by the same dongle without mutual interference and threat.

2. Some precautions

- Unless it is necessary, we do not recommend the use of multi-level directory to avoid chaos in management.
- In a directory, after a PIN verification reaches a certain security status, and then it switches to the parent directory or brother directory in the same layer, the security status of the original directory is lost. To enter the directory again requires PIN reverification to restore its previous security status.
- Although the storages of directories are separated by the "firewall", they still share memory spaces. To guarantee security, you must clear the contents in the memory by yourself after the codes are executed.

Sample Introduction

The program manipulates two directories respectively: it writes a Senselock EL executable file making forward bubble sorting into the directory "df01" (suppose the directory belongs to software A) and a Senselock EL executable file making backward bubble sorting to another directory "df02" (suppose this directory belongs to software B) and then invoke them respectively. It is clear that the two operations are mutually dependent and non-interfering.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below. Copy the successfully linked hex file *sample_11_hw.hex* and *sample_10_hw.hex* as used in the sample above to the directory of *..\sample_11\win32* and then debug and run the PC end program (To run the PC end program directly, the two hex files shall be placed in the same directory as the program). As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Senselock EL EXF Program

As mentioned above, what is written into the directory "df01" is a bubble sorting module like sample 10 and what is written into "df02" is a backward bubble sorting module that is slightly modified. The two modules in different directories are both named "ef11".

Detailed procedures omitted.

PC End Program

The function operation below deals with a directory and fulfills operations similar to sample 9: try to enter the target directory. If there exists such a directory, clear it; if not, create a new directory and then modify the developer PIN and user PIN under the directory. After the function is invoked, the directory stays at a security status of "developer level".

```
unsigned long OperateOnDir(
    unsigned char *lpszDirName,
    void *lpNewPin,
    unsigned long dwNewPinLen,
    void *lpNewUsrPin,
    unsigned long dwNewUsrPinLen)
{
    ...//
    /* attempt to change to destination directory */
    dwResult = S4ChangeDir(&stS4Ctx, lpszDirName);
    if (dwResult != S4_SUCCESS &&
        dwResult != S4_FILE_NOT_FOUND)
    {
        printf("Change directory failed!
                <error code: 0x%08x>\n", dwResult);
        return 0;
    }
    else if (dwResult == S4_SUCCESS)
    {
        ...
        /* dir exist , erase all its content */
        dwResult = S4EraseDir(&stS4Ctx, NULL);
        ...
    }
    else // destination directory not exists
    {
        ...
        /* create a directory named "lpszDirName"*/
        dwResult = S4CreateDir(&stS4Ctx, lpszDirName,
                               dwSubDirSize, S4_CREATE_SUB_DIR);
    }
    /* change DEV_PIN and USR_PIN after directory creation
        or erasion,that's important! */

    dwResult = S4ChangePin(&stS4Ctx, lpOldUsrPin,
            dwOldUsrPinLen, lpNewUsrPin, dwNewUsrPinLen,
            S4_USER_PIN);
    ...
    dwResult = S4ChangePin(&stS4Ctx,lpOldPin,dwOldPinLen,
            lpNewPin,dwNewPinLen,S4_DEV_PIN);
    return 1;
}
```

First invoke the above functions for “df01”, then write the forward bubble sorting module into the directory and invoke the module to process the input data:

```
if (!OperateOnDir("df01", lpNewPin, dwNewPinLen, lpNewUsrPin,
                  dwNewUsrPinLen))
{
...
}
dwResult = PS4WriteFile(&stS4Ctx, S4_EXE_FILE_ID,
            "sample_10_hw.hex", &dwFileSize,
           S4_CREATE_NEW, S4_HEX_FILE, &dwBytesWritten);
...
dwResult = S4VerifyPin(&stS4Ctx, lpNewUsrPin, dwNewUsrPinLen,
            S4_USER_PIN);
...
dwResult = S4Execute(&stS4Ctx, S4_EXE_FILE_ID, test, sizeof(test), test,
            sizeof(test), &dwBytesReturned);
```

After that, invoke the above function for “df02”, then write the backward bubble sorting module into the directory and invoke the module to process the input data:

```
if (!OperateOnDir("df02", lpNewPin_1, dwNewPinLen_1, lpNewUsrPin_1,
                          dwNewUsrPinLen_1))
{
...
```

```
}
dwResult = PS4WriteFile(&stS4Ctx, S4_EXE_FILE_ID,
             "sample_11_hw.hex", &dwFileSize, S4_CREATE_NEW,
            S4_HEX_FILE, &dwBytesWritten);
...
dwResult = S4VerifyPin(&stS4Ctx, lpNewUsrPin_1,
            dwNewUsrPinLen_1, S4_USER_PIN);
...
dwResult = S4Execute(&stS4Ctx, S4_EXE_FILE_ID,
             test, sizeof(test), test, sizeof(test),
&dwBytesReturned);
```

The operations on the two directories are fully independent and non-interfering in function and security. But this sample does not take into account the situation when software A where “df01” belongs and software B where “df02” belongs run simultaneously. If they run simultaneously, there might arise some problems. To make multiple independent software access the dongle simultaneously, it is necessary to consider the issue of process sharing. For details, please see sample 15.

Questions

Q1. Suppose independent hardware modules are stored in different subdirectories. Will the acquisition of the developer authorization for parent directory endanger the security of the contents in the subdirectory?

Sample 12: How to install/uninstall Senselock EL driver in your own codes?

Sample Gist

1. Ways of installing/uninstalling Senselock EL driver

   - While the developer is installing/uninstalling SDK, the device driver will be installed/uninstalled automatically.
   - The developer/software user runs *<SDK>\Driver\InstWiz3.exe* to install/uninstall the device driver.
   - If you are quite familiar with installation of device driver, you can choose to install/uninstall Senselock EL device driver manually. The *<SDK>\Driver\Obj* directory contains the files and guideline document required for manual installation of the driver.
   - Software developers write codes to installs/uninstalls the driver in their own programs.

   This sample introduces mainly the fourth approach. For the detailed introduction to the installation of the driver, please see "About Device Driver Installation" in Chapter 2 of *Senselock EL Developer Guide*.

Sample Introduction

This sample demonstrates how to install and uninstall Senselock EL in the codes.

Run *sample_12_win32.exe* and the latest driver of Senselock EL that has passed WHQL (Windows Hardware Quality Labs) will be installed in the default directory and all the old drivers will be deleted.

This program accepts the following input parameters in the meantime:

-u // uninstall all Senselock EL drivers (including all old versions) in the system

Attention: The following input parameters are used in the process of installation only:

-nc // keep drivers of previous version

-pcsc // install PC/SC compatible version

-rd [x] // count of installed virtual readers (The options *–pcsc* and *-rd [x]* are used only when Senselock EL is used as a network security device. If you use Senselock EL as dongle only, there is no need to use these options)

-path [string] // specify the installation directory for the driver. If the parameter is not specified, the driver will be installed by default under the directory of *"<%SYSTEM_DRV%>\Program Files\Senselock\Driver"*.

Sample Implementation

It is necessary to include *sample_12\win32\mkSetup.lib* in the VC project and make sure the target executable program *sample_12_win32.exe* can access *mkSetup.dll*.

PC End Program

Include header file:

```
#include "s4drv.h"
```

The code below processes the input parameters and decides, according to some parameters, whether to invoke the installation or uninstallation of the driver:

```
HandleArgment(char *argv, int argc);/* handle argument */
...//
if(isInst)
{
    printf("start to install Senselock Drivers....\n");
    /* install driver */
    dwRet = s4drv_Install(lpszPath, dwReaderCount, dwFlag);
    ...// check dwRet
}
else
{
    printf("start to uninstall Senselock Drivers....\n");
    /* uninstall driver */
    dwRet = s4drv_Uninstall();
    ...// check dwRet
}
printf("finish\n");
```

The last invoked function checks whether the system needs rebooting after the driver installation/uninstallation is completed.

```
if(s4drv_IsNeedReboot())
    printf("you must restart your system!");
```

Questions

Sample 13: How to realize dongle insert/remove notification?

Sample Gist

1. How to notify the windows program when the dongle is inserted/removed?

Sometimes we might hope to receive Senselock EL dongle insert/remove notification in windows program. This sample demonstrates how to fulfill this objective.

2. Attention:

In this sample, the ways some functions are used (such as device notification registration function, HID registration function) are not supported in win95 and NT4.0. If they must be used, they need to be modified according to the corresponding function definitions and function libraries in Platform SDK and Windows DDK. For details, please see the description in MSDN.

Sample Introduction

This sample demonstrates how to receive Senselock EL insert/remove notification while composing a windows program.

After the program is run, click the "Register" button to register Senselock EL "Device Change Notification" with the current window. In this way, when a device is inserted/removed, there will pop up "message boxes" respectively: "Senselock EL inserted" and "Senselock EL removed". Click the "Unregister" button and there will not be message box any more.

Sample Implementation

PC End Program

Create a simplest dialog-based windows program using "Create New Project Wizard" in VC6.0. Add a button and name it as "`IDC_REGISTER`" and then add a button-clicking handler member function "`OnRegister ()`".

In case no Platform SDK is installed, it is necessary to add the following sentences to `<stdafx.h>`:

```
#ifndef WINVER
#define  WINVER 0x0500
#endif
```

Add the following header file and variable definition to *sample_13Dlg.cpp*:

```
#include <dbt.h>
...//
// function prototype for getting HID GUID
typedef void (__stdcall *FUNC_TYPE_HIDGUID) (
            OUT   LPGUID   HidGuid
            );
FUNC_TYPE_HIDGUID _HidD_GetHidGuid = NULL;

// Senselock EL USB version GUID
static GUID GUID_CLASS_ITOKEN2 =
{0x171638f7, 0x1ead, 0x4873, {0xba, 0x98, 0xc9, 0x66, 0xab, 0xcf, 0x1,
0x42}};
static GUID GUID_CLASS_ITOKEN2_HID;

BOOL    bRegisted  = FALSE;
HDEVNOTIFY hDevNotify1 = NULL;
HDEVNOTIFY hDevNotify2 = NULL;
```

Click “Register/Unregister” to invoke the following code to register/unregister Senselock EL “Device Change Notification” with the current window. Notice that at the time of registration, the “device interfaces” used are GUID (USB mode) of Senselock EL and GUID (HID mode) of HID device. As for the specific meaning and usage of some windows system API functions used in the codes below, please refer to MSDN.

```
void CSample_13Dlg::OnRegister()
{
   ...//
    {
   // Get HID device GUID
   _HidD_GetHidGuid ();
    }
   ...//
    if (!bRegisted) // if not registered
    {
          // Register ITOKEN2 device notification
          hDevNotify2 = RegisterDeviceNotification( hWnd,
              &dbf,DEVICE_NOTIFY_WINDOW_HANDLE);
          ...//
          // Register ITOKEN2 HID device notification
          hDevNotify1 = RegisterDeviceNotification( hWnd,
              &dbf,DEVICE_NOTIFY_WINDOW_HANDLE);
    }
    else // registered, start unregistering…
    {
         if (hDevNotify1)
        // Unregister ITOKEN2 device notification
         {
             bRes = UnregisterDeviceNotification(hDevNotify1);
             ASSERT(bRes);
         }
         if (hDevNotify2)
        // Unregister ITOKEN2 HID device notification
         {
             bRes = UnregisterDeviceNotification(hDevNotify2);
             ASSERT(bRes);
        }
     ...//
     }
}
```

After “Device Change Notification” is successfully registered, it is necessary to add the following sentence to the MFC message mapping part:

```
BEGIN_MESSAGE_MAP(CSample_13Dlg, CDialog)
    //{{AFX_MSG_MAP(CSample_13Dlg)
    ...//
    ON_WM_DEVICECHANGE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

After that, add the following member function. Whenever a Senselock EL device is inserted/removed, it will be invoked. Attention: the plugging of other plug-and-play hardware devices (e.g. USB mobile hard disk, USB mouse, etc.) will also trigger off the invocation of the function. Therefore, the function will determine whether the current operation is related to Senselock EL device based on the message parameters (mainly the GUID of the device).

```
BOOL CSample_13Dlg::OnDeviceChange(UINT nEventType, DWORD dwData)
{
    ...//
    switch(nEventType)
    {
    case DBT_DEVICEARRIVAL:
        ...//
        /* Check GUID to see whether it's a Senselock EL device.
```

```
        Notice:the GUIDs of HID devices are of the same,so
        we have to make further check on PID and VID */

        if (!memcmp(&pDbt->dbcc_classguid,
             &GUID_CLASS_ITOKEN2, sizeof(GUID)) ||
                                    (!memcmp(&pDbt->dbcc_classguid,
              &GUID_CLASS_ITOKEN2_HID, sizeof(GUID)) &&
             !memicmp(&pDbt->dbcc_name, hidDevPath,
              strlen(hidDevPath))))
        {
            // TODO: Add your handler code here
        }
        break;
    case DBT_DEVICEREMOVECOMPLETE :
         ...// check to see whether is a Senselock EL device,
            // if it is,do the following:
        {
            // TODO: Add your handler code here
        }
        break;
    default:
        break;
    }
    return TRUE;
}
```

Questions

Sample 14: How to operate device in multithread?

Sample Gist

1. How to operate device in multithread?

After the Senselock EL device is opened in "share" mode, the driver program at the bottom layer can ensure independence in sending commands to the device each time. However, when other threads switch the directory of Senselock EL device, the security status of the operating directory of the original thread will be lost. When it is switched back once again, it might impact the subsequent operations. In view of this, it should be guaranteed that after a thread verifies the PIN and before it executes the command that requires corresponding security status, the operation will not be interrupted.

Sample Introduction

This sample demonstrates how to operate Senselock EL device in multi-thread. To highlight priorities, our program is based on the successful completion of sample 11. In this case, suppose Senselock EL dongle has created successfully two directories: "df01" and "df02" and written into them successfully the forward bubble sorting executable file and the backward bubble sorting executable file. The ID of the two files is "ef11".

In the sample, two operating sub-threads have been created. The job for the thread function is quite simple: switch to the corresponding operating directory and invoke the executable program under the directory to process the input data. This operation sequence shall be done 100 times.

Sample Implementation

Based on the successful completion of sample 11, run PC end program. It is clear that the two thread operations are mutually independent and non-interfering.

PC End Program

To avoid the situation where the operating sequence of one thread is interrupted by another thread in the middle, we achieve thread synchronization using kernel object "Mutex" of the operating system. Insert the operating sequence code (including verifying PIN, a full array of operations like running Senselock EL program, etc.) in between the following codes of the thread function:

```
HANDLE hMutex;

hMutex = CreateMutex(
            NULL,               // no security descriptor
            FALSE,              // mutex not owned
            "NameOfMutexObject");  // object name
...//Check hMutex

WaitForSingleObject(hMutex, INFINITE);

...// Operating sequence

if (! ReleaseMutex(hMutex))
{
    return 0;
}
```

After the sub-thread has done all the work, press any key to exit the main thread of the program.

Questions

Q1. How to open the device in “exclusive”mode?

Sample 15: How to operate device in multiprocess?

Sample Gist

1. How to operate device in multiprocess?

After the Senselock EL device is opened in "share"mode, the driver can ensure independence in sending commands to the device each time. However, when other process switch directories, the security status of the operating directory of the original process will be lost. When it is switched back once again, that might impact the subsequent operations. In view of this, it should be guaranteed that after a process verifies the PIN and before it executes the command that requires corresponding security status, the operation will not be interrupted.

Sample Introduction

This sample demonstrates how to operate Senselock EL device in multi-process. To highlight priorities, our program is based on the successful completion of sample 11. In this case, suppose Senselock EL dongle has created successfully two directories: "df01" and "df02" and written them successfully into the forward bubble sorting executable file and the backward bubble executable file. The ID of the two files is "ef11".

In the sample, two programs have been created. The job for each program is quite simple: switch to the corresponding operating directory and invoke the executable program under the directory to handle the input data. This operation sequence shall be done 100 times. In the meantime, run the two programs to accomplish their respective work.

Sample Implementation

Based on the successful completion of sample 11, run PC end program*sample_15_p1.exe* and *sample_15_p2.exe*. It is clear that the two process operations are mutually independent and non-interfering.

PC End Program

To avoid the situation where one operation sequence of one process is interrupted by another process in the middle, we achieve process synchronization using kernel object "Mutex"of operating system. Insert the operation sequence code of a program (including verifying PIN, a full array of operations like running Senselock EL program, etc.) in between the following codes:

```
HANDLE hMutex;

hMutex = CreateMutex(
            NULL,               // no security descriptor
            FALSE,              // mutex not owned
            "NameOfMutexObject");  // object name
...//Check hMutex

WaitForSingleObject(hMutex, INFINITE);

...// Operating sequence

if (! ReleaseMutex(hMutex))
{
    return 0;
}
```

Questions

Q1. Under what situation is a problem very likely to occur if Senselock EL is operated in multi-thread/multi-process but without synchronization mechanism?

3. About Code Transplant

This chapter introduces how to transplant codes into Senselock EL device, involving basic operation methods, operation processes, precautions and some techniques in using Senselock EL.

Sample 16: How to accomplish input/output using communication buffer?

Sample Gist

1. Take into account the issue of data alignment between two communication parties and avoid data interpretation error.

2. Take into account the issue of byte sequence conversion and avoid data interpretation error.

3. Avoid buffer transgression.

For the introduction to data alignment, byte sequence conversion and communication buffer border, please see Chapter 4 of *Senselock EL Developer Guide*.

4. What is TLV format? What are the benefits of using TLV format?

TLV is short for three words: Tag, Length and Value, whose structure is as follows:

Tag	Length	Value

TLV format can clearly identify data objective and length, very convenience in processing data of variable length.

5. How to organize user communication data with structure?

When the input/output data contain many data items, they can be organized by defining a structure to make it easy to process.

6. PC end codes and Senselock EL hardware codes share data-definition header file.

When PC end software code and hardware internal code transmit data using a structure, the structure can be defined in an independent header file for the convenience in maintaining codes.

Attention: When data are transported between PC code and hardware internal code, avoid using “int” type, the reason being that they define “int” differently. In PC “int” is usually the same as “long” while in Senselock EL code “int” is equivalent to “short”.

7. General ways to handle internal running errors of Senselock EL:

   - An independent function can be defined to handle internal running errors.
   - The prompt messages in error handling shall be used mainly for program debugging in the middle of development. When the software is officially released, do not return error prompt messages so as not to leave any clue to the cracker!

Sample Introduction

This sample demonstrates how to transmit formatted input/output data between PC end and Senselock EL device using the communication buffer. Since the sample is mainly designed to illustrate how to do the most fundamental and important jobs like input/output, for the sake of convenience, we choose quite a simple algorithm: Work out the volume of a cylinder or a cuboid. The EXF file written into Senselock EL is:

Input: sub-function number, identify whether to work out the volume of a cylinder (CMD_CYD = 0x01) or a cuboid (CMD_CUB = 0x02). If it is a cylinder, input the underside radius and height of the cylinder; if it is a cuboid, input the length, width and

height of the cuboid.

Output: volume value.

For example, if the following is input:

Subfunction tag. = CMD_CYD; // Work out the cylinder's volume

Underside radius ＝ 2.3;

Cylinder height ＝ 3.5;

If π = 3.14159 is used in EXF, the volume of the cylinder can be worked out like this: 58.16653

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_16_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef16" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide.*

Public Header File

In this sample, to make it easy to input/output data between PC end and Senselock EL hardware program, we have defined as follows a structure of TLV format called `IO_PACKAGE` in *sample_16.h* where T (Tag) indicates flag byte, usually designed to indicate sub-function number at the time of input and return an error code at the time of output. L (Length) indicates the length of the input/output data and V (Value) indicates input/output data. They correspond respectively to the three structure members `tag`, `len`, and `buff` in the structure below. Both PC end and Senselock EL EXF source file use `#include` *sample_16.h.*This makes easy data conversion and information extraction between them.

```
/* IO package */
typedef struct _IO_PACKAGE
{
    unsigned char tag;
    unsigned char len;
    unsigned char buff[MAX_IO_DATA_SIZE];
} IO_PACKAGE;
```

Before making structure type statement, we add compiler directive `#pragma pack(push, 1)` to force the compiler to align by 1 byte. After the statement is made, add `#pragma pack(pop)` to restore the original setting. As for the reason why data alignment is required, please see Chapter 4 "Alignment Issue" of *Senselock EL Developer Guide.*

After data alignment is adopted, the members of the structure: `tag` and `len` are of fixed-length, each seizing one byte, with a total of 2 bytes. We have defined a macro `IO_PACKAGE_HEADER_SIZE` to indicate this value. In addition, due to the limit of the internal resources of Senselock EL, the communication buffer shall not exceed 250 bytes in size. In other words, the data transported from PC to Senselock EL or from Senselock EL to PC each time shall not exceed 250 bytes in size. With the two bytes seized by the TLV structure packet header deducted, the maximum data size that the structure can carry is 250-2=248 (0xf8). We have defined a macro `MAX_IO_DATA_SIZE` to indicate this value.

```
#define IO_PACKAGE_HEADER_SIZE 2 //IO package header length:tag+len
#define MAX_IO_DATA_SIZE   0xf8 // maximum IO data size
```

The above-mentioned TLV structure features versatility. It is enough for cases with simple input/output. However, when the input/output data packet contains many parameter items, it is still not convenient to use this TLV structure only. For example, in this sample, the data parameters

that need to be input include length, width, height, downside radius, etc. The data self-defined by users can be sequenced like this in the memory:

length	width	height

We can copy one by one the user-defined data to the input buffer using `memcpy()` or similar memory operation functions, but this will cause some difficulties in the understanding and maintenance of the program code. A better solution is to define a "user-defined data" structure to assign value one by one to its members before it is invoked and finally copy the whole structure to the input communication buffer.

To this end, we have introduced several "user-defined" structures as shown below to represent the cylinder and cuboid:

```
/* user-specific data block */
typedef struct _CYLINDER
{
    float radius;
    float height;
} CYLINDER;

typedef struct _CUBOID
{
    float length;
    float width;
    float height;
} CUBOID;
```

In addition, we have also defined some universal error codes to indicate the execution results of Senselock EL EXF. You can add your own specific error codes here.

```
#define ERR_SUCCESS             0x00    //  EXF Execution Success
#define ERR_SES                 0x80    //  SES Function Error
#define ERR_INVALID_PARAMETER   0x81    //  Invalid Parameter
```

Finally, the complete *sample_16.h* is like this:

```
#ifndef __SAMPLE_16_H__
#define __SAMPLE_16_H__

#ifdef _WIN32
#pragma pack(push, 1)
#endif

#define IO_PACKAGE_HEADER_SIZE 2 //IO package header length:tag+len
#define MAX_IO_DATA_SIZE   0xf8 // maximum IO data size

/* common error code. */
#define ERR_SUCCESS             0x00    // success
#define ERR_SES                 0x80    // SES function execution error
#define ERR_INVALID_PARAMETER   0x81    // invalid parameter

/* user-specific data block */
typedef struct _CYLINDER
{
    float radius;
    float height;
} CYLINDER;

typedef struct _CUBOID
{
    float length;
    float width;
    float height;
} CUBOID;

/* IO package */
```

```
typedef struct _IO_PACKAGE
{
    unsigned char tag;
    unsigned char len;
    unsigned char buff[MAX_IO_DATA_SIZE];
} IO_PACKAGE;

#ifdef _WIN32
#pragma pack(pop)
#endif
```

Senselock EL EXF Program

Global variable that might be used in the EXF program is defined below:

```
/* global variable defination. */
unsigned char last_ses_error = 0; /* last SES error */
IO_PACKAGE out_pkg;  /* output package */
IO_PACKAGE *output = &out_pkg;
IO_PACKAGE *input = (IO_PACKAGE *)pbInBuff; /* input package */
```

After receiving the input buffer from PC end to the memory area pointed by *pbInBuff* and converting them using the pointer below, Senselock EL EXF can easily process each parameter in the input data packet.

```
IO_PACKAGE *input = (IO_PACKAGE *)pbInBuff; /* input package */
```

The code below judges the sub-function number first, makes “user-defined” structure pointer conversion accordingly based on different sub-function numbers, then converts the byte sequence of the input multiple-byte parameters and finally invokes the corresponding sub-functions. As for the reason why byte sequence conversion is required, please see Chapter 4 “Big-Endian and Little-Endian” of *Senselock EL Developer Guide*.

```
switch (input->tag)
{
    case CMD_CYD: // calculate cylinder's volume
    {
            // for convenience in handling user specific data block
            CYLINDER *dblk = (CYLINDER *)input->buff;
            float radius = dblk->radius;
            float height = dblk->height;
            // from PC's Little Endian to C-Compiler's settings
            LE32_TO_CC(&radius);
            LE32_TO_CC(&height);
            // call sub-function
            result = cylinder_volume(radius, height);
    }
    break;
    case CMD_CBD: // calculate cuboid's volume
        {
            CUBOID *dblk = (CUBOID *)input->buff;
            float length = dblk->length;
            float width = dblk->width;
            float height = dblk->height;
            LE32_TO_CC(&length);
            LE32_TO_CC(&width);
            LE32_TO_CC(&height);
            result = cuboid_volume(length, width, height);
        }
    break;
    default:
            error_exit(ERR_INVALID_PARAMETER, NULL, 0);
    break;
}
```

The code below fulfills the simple algorithm as mentioned afore: work out the volume of the cylinder and cuboid.

```
float cylinder_volume(float radius, float height)
{
    return Pi*radius*radius*height;
}

float cuboid_volume(float length, float width, float height)
{
    return length*width*height;
}
```

As the data `result` to be returned is multi-byte data, it is necessary to submit it to byte sequence conversion before it is returned:

```
CC_TO_LE32(&result);
```

After that, we set the *output* packet. Please notice that after invoking almost all the SES functions (except `_mem_cmp()` and most of the computational functions), do remember to check the returned value immediately, such as the check on the execution result of SES function `_mem_copy()` as shown below. If something goes wrong, invoke error exit codes.

```
ret = _mem_copy(&output->buff, &result, sizeof(result));
if (ret != SES_SUCCESS) /* Almost every SES execution must be checked
*/
{
       last_ses_error = ret;
       error_exit(ret, &last_ses_error, 1);
}
length += sizeof(result);
output->tag = ERR_SUCCESS;
output->len = length;
```

Finally, set the communication buffer and exit:

```
_set_response(IO_PACKAGE_HEADER_SIZE+output->len, (unsigned
              char *)output);
_exit();
```

The function below will be invoked when Senselock EL EXF encounters error in execution. It will complete the setting of returned value and exit the EXF program.

```
/*
    Exit upon an error.

    Parameters:
        errcode[in]: error code
        msg[in]: additional error message
        msglen[in]: message length

    Return:
        none.

    Remarks:
        none.
*/
void error_exit(unsigned char errcode, unsigned char *msg, int msglen)
{
    output->tag = errcode;
    output->len = 0;

    /* for debug purpose only */
#ifdef  _DEBUG
    if (msglen != 0 && msg != NULL)
    {
        memcpy(output->buff, msg, msglen);
        output->len = msglen;
    }
#endif
```

```
    _set_response(2+output->len, (unsigned char *)output);
    _exit();
}
```

Download the successfully linked result *sample_16\hardware\sample_16_hw.hex* to the root directory of Senselock EL device and name it "ef16" and then you can run PC end program properly.

PC End Program

Next let's take a look at the PC end program. On top of the sample as shown previously, *sample_16_win32.c* adds the following two global variables to process input/output data packets.

```
IO_PACKAGE stDataPkgIn = {0};
IO_PACKAGE stDataPkgOut = {0};
```

stDataPkgIn.tag is generally designed to indicate the sub-function number to be invoked; *stDataPkgIn.len* is designed to indicate the length of the data to be input into Senselock EL device; *stDataPkgIn.buff* is filled with the data to be input and processed.

We have defined a function `cylinder_volume()`: the Senselock EL device can be invoked to work out the volume of the cylinder. Given below is the implementation of it. As shown below, the "user-defined" structure `CYLINDER` in the public header file is used:

```
CYLINDER stDataBlk = {0};
```

Given below is a description of how to fill out this structure:

```
stDataBlk.radius = radius;
stDataBlk.height = height;
```

Given below is a description of how to fill out the *stDataPkgIn* structure to input data:

```
stDataPkgIn.tag = CMD_CYD;
stDataPkgIn.len = sizeof(CYLINDER);
memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
```

After it is filled out, the input buffer memory at the PC end might be sequenced like this:

stDataPkgIn.tag = CMD_CYD	stDataPkgIn.len = 8	stDataBlk.radius = 2.3 (4 byte float)	stDataBlk.height = 3.5 (4 byte float)

After `S4Execute()` is invoked, *stDataPkgOut.tag* stores the execution error code of Elite EXF; *stDataPkgOut.len* stores the returned data length; *stDataPkgOut.buff* stores the output data. When it returns under normal situation, the output buffer memory at the PC end might be sequenced like this:

stDataPkgOut.tag	stDataPkgOut.len	buff[0]	…	buff[len-1]	…

The implementation code of the full sub-function `cylinder_volume()` at the PC end is like this:

```
unsigned long cylinder_volume(float radius, float height, float
*p_result)
{
    CYLINDER stDataBlk = {0};
    /* asssemble input package */
    stDataBlk.radius = radius;
    stDataBlk.height = height;

    stDataPkgIn.tag = CMD_CYD;
    stDataPkgIn.len = sizeof(CYLINDER);
    memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
```

```
    printf("Calling Senselock EL EXF %s...\n", S4_EXE_FILE_ID);
    dwResult = S4Execute(&stS4Ctx, S4_EXE_FILE_ID,
              &stDataPkgIn,
                IO_PACKAGE_HEADER_SIZE + stDataPkgIn.len,
              &stDataPkgOut, sizeof(IO_PACKAGE),
             &dwBytesReturned);
   if (!CheckS4ExecutionResult(dwResult, dwBytesReturned,
                           stDataPkgOut.tag))
    {
        return 0;
    }

   memcpy(p_result, stDataPkgOut.buff, stDataPkgOut.len);
   return 1;
}
```

In the program segment above, we have invoked the following several functions to process the returned results of Senselock EL EXF.

Important Reminder: The main purpose of using these functions is to debug. After the program is released officially, the codes with similar functions shall be removed to avoid leaving any error prompt message valuable to the cracker!!! In the program segment as shown below, a compiler directive is used to control error output in a non-debug state.

```
/* for debug purpose only : handle results returned by stDataPkgOut.tag
*/
#ifdef _DEBUG
void HandleError(unsigned char bErrCode, unsigned char *pbErrInfo)
{
    switch(bErrCode)
    {
    case ERR_SUCCESS:
        printf("Success!\n");
        break;
    case ERR_SES:
        printf("Error during SES Execution :\n%s\n",
               SesErrMsgs[*pbErrInfo]);
        break;
    case ERR_INVALID_PARAMETER:
        printf("Invalid input,pls check your input data\n");
        break;
    default:
        printf("Undefined error code!\n");
        break;
    }
    return;
}
#else
void HandleError(unsigned char bErrCode, unsigned char *pbErrInfo)
{
    return;
}
#endif
```

After invoking `S4Execute()` or `S4ExecuteEx()`, the function below is executed to check the running result and handle errors. There are three kinds of errors that might occur:

- Senselock EL API `S4Execute()` returns an error;
- Senselock EL API `S4Execute()` is executed successfully, but the returned value of the last parameter *lpBytesReturned* of the API function is < 2. That is to say, the length of the data returned from the communication buffer is <

IO_PACKAGE_HEADER_SIZE. This indicates there might be a logical problem or execution error in the composition of Senselock EL EXF code;

- Senselock EL API `S4Execute()` is executed successfully, but the output data packet *stDataPkgOut.tag* returned by the `S4Execute()` function is not equal to `ERR_SUCCESS`. This indicates Senselock EL EXF is not completed successfully. At this point, *stDataPkgOut.buff* contains more error information.

```
/* check result for S4Execute() */
unsigned long CheckS4ExecutionResult(unsigned long
                    dwResult,unsigned long dwBytesReturned,
                     unsigned char tag)
{
    if (dwResult != S4_SUCCESS)
    {
        printf("S4 Execution failed! <error code: 0x%08x>\n",
               dwResult);
        return 0;
    }

    if (dwBytesReturned < 2)
    {
        printf("Invalid data package returned!\n");
        return 0;
    }

    if (tag != ERR_SUCCESS)
    {
        HandleError(stDataPkgOut.tag, stDataPkgOut.buff);
        return 0;
    }
    return 1;
}
```

Questions

Q1. What are the disadvantages of transporting data with structure?

Q2. If the structure in this sample is not used, how to handle input/output to accomplish the same purpose?

Q3. Under what situation is byte sequence conversion required?

Sample 17: How to use code memory?

Sample Gist

1. What is Code Memory?

The memory space seized by the code segment of Senselock EL EXF is called code memory. As for the detailed description of the memory structure of Senselock EL, please see Chapter 4 "Memory Structure" of *Senselock EL Developer Guide.*

2. Why do we use code memory? Under what situation is it better to use code memory?

Code memory shows benefits as follows:

- The data defined in the code memory are like those defined in the internal file, but they are never as troublesome as file operations.
- It is as easy to operate them as to operate the memory, and it does not occupy the valuable RAM of the device.
- The maximum code memory of Senselock EL is up to 32K, much greater than the RAM (about 2K bytes) of the Senselock EL device.

Therefore, to make best use of the code memory is of great help to exert fully the performance of Senselock EL device. We can often use it to define some big read-only constant tables, character strings, etc.

Sample Introduction

This sample demonstrates how to store a read-only 2D table in the Code Memory and make inquiry operations in the code. For the convenience of description, let us suppose the storage is a multiplication table. The EXF functions written into Senselock EL are as follows:

Input: line number and column number

Output: query result

For example, if the following is input:

```
stDataPkgIn.len = 2;
stDataPkgIn.buff[0] = 8; //line
stDataPkgIn.buff[1] = 9; //column
```

The result can be worked out through the query of the table: 72

For the description of how to use the TLV structure `stDataPkgIn`, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_17_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef17" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide.*

Senselock EL EXF Program

The code below defines a multiplication table in the code memory:

```
unsigned char code table[9][9] =
{
 1,   2,   3,   4,   5,   6,   7,   8,   9,
 2,   4,   6,   8,  10,  12,  14,  16,  18,
 3,   6,   9,  12,  15,  18,  21,  24,  27,
 4,   8,  12,  16,  20,  24,  28,  32,  36,
 5,  10,  15,  20,  25,  30,  35,  40,  45,
 6,  12,  18,  24,  30,  36,  42,  48,  54,
 7,  14,  21,  28,  35,  42,  49,  56,  63,
 8,  16,  24,  32,  40,  48,  56,  64,  72,
 9,  18,  27,  36,  45,  54,  63,  72,  81
};
```

After the table is defined, the following code can be used to make table query operation on the input data and assemble the result into output data packet:

```
unsigned char line = input->buff[0]-1;
unsigned char column = input->buff[1]-1;
output->buff[0] = table[line][column]; /* lookup in table */
output->tag = ERR_SUCCESS;
output->len = sizeof(char);
```

As for how to format the output data packet using the pointer *output* pointing to the TLV structure, please see sample16.

Download the successfully linked result *sample_17\hardware\sample_17_hw.hex* to the root directory of Senselock EL device and name it "ef17".

PC End Program

Assemble the input data packet

```
stDataPkgIn.tag = 0; /* no meaning */
stDataPkgIn.len = 2; /* length of input data */
stDataPkgIn.buff[0] = 8; /* line */
stDataPkgIn.buff[1] = 9; /* column */
...//Call S4Execute() and check result
```

If it is successful, process the output data packet.

```
printf("The result is: %d\n", stDataPkgOut.buff[0]);
```

As for the description of how to format input/output data packet using TLV structures *stDataPkgIn* and *stDataPkgOut*, please see sample16.

Questions

Q1. What are advantages and disadvantages of using Senselock EL code memory in comparison with Senselock EL data files?

Q2. In terms of some specific applications, what kind of data can be put into Senselock EL code memory?

Sample 18: How to use shared memory?

Sample Gist

1. What is shared memory?

We use the memory in Senselock EL that is "not cleared automatically" as "shared memory". For the detailed description for the memory structure of Senselock EL, please see Chapter 4 "Memory Structure" of *Senselock EL Developer Guide.*

2. Why do we use shared memory? Under what situation is it better to use shared memory?

There are several typical uses of shared memory:

- If the input data are too large in size (more than 250 bytes), they can be input by several times and stored in the shared memory. After they are all inputted, they are subject to processing at once.
- Some processing requires several steps. The processing steps that follow can use the result left in the memory by the preceding processing steps.
- Algorithm result can be shared between several EXFs.
- Makes several calls which have mutual relations to the Senselock EL device so as to prevent against the universal sharing device of the dongle.

3. What precautions shall be taken in the use of the shared memory?

- The upper-end of the memory area close to the border is usually specified as shared memory to prevent its being covered by variable/stack in the execution of other EXFs.
- If the dongle is going to be shared with other developers, clear sensitive information in the memory as much as possible using `_mem_set()`.

Sample Introduction

This sample demonstrates how to buffer data between times of invocation using the shared memory. The EXF functions written into Senselock EL are as follows:

Input: When the sub-function number is (CMD_RESET = 0x00), clear the variable in the shared memory. When it is (CMD_COUNT = 0x01), input the single subject scores of the primary school pupils by batch; when it is (CMD_GET), take the statistic result.

Output: the number of scores in the following four score segments: 90 or above (including 90), 80~90 (including 80), 60~80 (including 60), below 60.

For example, the first operation is to clear the variable in the shared memory:

```
    stDataPkgIn.tag = CMD_RESET; /* zerorize the share memory
*/

    stDataPkgIn.len = 0;
```

The second operation is to input 5 scores:

```
    stDataPkgIn.tag = CMD_COUNT;

    stDataPkgIn.len = 5;

    stDataPkgIn.buff[0] to [4] are 99, 87, 65,  55, 60;
```

The third operation is to input 5 scores:

```
stDataPkgIn.tag = CMD_COUNT;

stDataPkgIn.len = 5;

stDataPkgIn.buff[0] to [4] are 88, 82, 67, 78, 76;
```

The fourth operation is to get the result. The number of scores in the four score segments is respectively 1, 3, 5 and 1.

For the description of how to use the TLV structure *stDataPkgIn*, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_18_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef18" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Senselock EL EXF Program

The code below defines four variables in the shared memory area:

```
DEFINE_AT(unsigned char, less_than_60, 0x400, RAM_EXT);
DEFINE_AT(unsigned char, btw_60_80, 0x401, RAM_EXT);
DEFINE_AT(unsigned char, btw_80_90, 0x402, RAM_EXT);
DEFINE_AT(unsigned char, larger_than_90, 0x403, RAM_EXT);
```

When the sub-function number is (`CMD_RESET = 0x00`), invoke `reset_share_var()` to clear variables in the shared memory. Given below is the implementation code of the function:

```
void reset_share_var()
{
    less_than_60 = 0;
    btw_60_80 = 0;
    btw_80_90 = 0;
    larger_than_90 = 0;
}
```

When it is (`CMD_COUNT = 0x01`), input the single subject scores of primary school pupils by batch. Given below is the process of making statistics of the input data:

```
for (; i< input->len; i++)
{
  if (*(input->buff + i) > 100)
  {
      error_exit(ERR_INVALID_PARAMETER, NULL, 0);
  }
  if (*(input->buff + i) >= 90)
  {
      larger_than_90++;
  }
  else if ((*(input->buff + i) < 90) && (*(input->buff + i)
                        >= 80))
  {
      btw_80_90++;
  else if ((*(input->buff + i) < 80) && (*(input->buff + i)
                        >= 60))
  {
      btw_60_80++;
  }
```

```
  else if (*(input->buff + i) < 60)
  {
      less_than_60++;
  }
}
```

When it is (CMD_GET), extract the result from the shared memory and assemble it to the *output* data packet:

```
output->buff[0] = larger_than_90;
output->buff[1] = btw_80_90;
output->buff[2] = btw_60_80;
output->buff[3] = less_than_60;
```

As for how to format output data packet using *output*, the pointer to the TLV structure, please see sample16.

Download the successfully linked result *sample_18\hardware\sample_18_hw.hex* to the root directory of Senselock EL device and name it "ef18".

PC End Program

In the first invocation, clear the variables in the shared memory:

```
stDataPkgIn.tag = CMD_RESET; /* initiate the share memory */
stDataPkgIn.len = 0;
... // call S4Execute() and check result
```

In the second invocation, input the scores of the first group of five pupils:

```
stDataPkgIn.tag = CMD_COUNT;
stDataPkgIn.len = 5;
stDataPkgIn.buff[0] = 99;
stDataPkgIn.buff[1] = 87;
stDataPkgIn.buff[2] = 65;
stDataPkgIn.buff[3] = 55;
stDataPkgIn.buff[4] = 60;
... // call S4Execute() and check result
```

In the third invocation, input the scores of the second group of five pupils:

```
stDataPkgIn.tag = CMD_COUNT;
stDataPkgIn.len = 5;
stDataPkgIn.buff[0] = 88;
stDataPkgIn.buff[1] = 82;
stDataPkgIn.buff[2] = 67;
stDataPkgIn.buff[3] = 78;
stDataPkgIn.buff[4] = 76;
... // call S4Execute() and check result
```

In the fourth invocation, extract statistic data from the shared memory:

```
stDataPkgIn.tag = CMD_GET;
stDataPkgIn.len = 0;
... // call S4Execute() and check result
```

If it is successful, process the output data packet and print the statistic results.

For the description of how to format the input/output data packet using the TLV structures: *stDataPkgIn* and *stDataPkgOut*, please see sample16.

Questions

Q1. Input data more than 250 bytes by several times using the shared memory. Under what situation can this feature be used?

Q2. Senselock EL EXF under different directories (or the same directory) can also use the shared memory. Under what situation can this feature be used?

Q3. How to achieve the goal of making several calls to Senselock EL which are mutually related using the shared memory?

Sample 19: How to use hardware function library?

Sample Gist

1. What is hardware function library?

The hardware function library is similar in concept and usage to the static library as used in PC end programming.

2. Why do we use hardware function library?

Similar to the static library as used in the PC end programming, it can place functions of the same category in a static library for the convenience of organizing and use.

Sample Introduction

This sample demonstrates how to compose Senselock EL hardware function library and how to use it. For the convenience of description, we continue to use the bubble sorting algorithm as used in the previous sample.

Input: disordered array

Output: sorted array

Sample Implementation

First create a hardware function library with Keil and then add the successfully created function library *sample_19_lib.LIB* to the Keil project in this sample. After that, download the successfully linked hex file *sample_19_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef19" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide.*

Create function library in Keil

It is essential to create an independent project. The creating steps, choice of chip type and memory model are very similar with the creation of a general Senselock EL executable program project, the only difference being that "Create Library" rather than "Create Executable" is selected in Target|Options|Output. After the function library is successfully created, add the created function library into the executable project. For the details on creating general Senselock EL project in Keil, please see Chapter 5 of *Senselock EL Developer Guide.*

Senselock EL EXF Program

The function implemented in the function library is invoked in the same way as functions declared in the same executable project. What is only required is to declare the original form of the invoked function as follows prior to calling it:

```
void bubble_sort(unsigned char *mydata ,int len);
```

Other details omitted.

Download the successfully linked result *sample_19\hardware\sample_19_hw.hex* to the root directory of Senselock EL device and name it "ef19".

PC End Program

Omitted

Questions

Q1. What problems might arise if different memory models are selected for the hardware function library and the Senselock EL executable program?

4. File Operations

This chapter introduces how to operate the internal files of Senselock EL using the system extension service (SES) function, including basic operations like creating and reading & writing a file, the operating process, precautions and some tips.

Sample 20: How to read & write a Senselock EL data file?

Sample Gist

1. How to read & write a Senselock EL data file?

The data file is designed only to store binary data. The program on the PC cannot read data files directly from Senselock EL. To read the data files in Senselock EL, you must use the EXF in Senselock EL. You can write files directly using `S4WriteFile()`, but this requires developer level authorization. To write data files under the user level authorization, you also need to use the EXF in Senselock EL.

2. Watch out the security issue when operating files. Using the sample directly will cause many hidden security troubles!

For the convenience of description, this sample does not encrypt any written data. In addition, the data files in the dongle are output directly to the PC end. This will cause great hidden security trouble. In the actual use, the input data shall be encrypted (just like what is used on our remote-update platform solution). Moreover, the use of the content of the read files, unless encrypted, shall be strictly restricted to the dongle.

What is more, this sample contains the "developer level" operation codes. In the release of the software, be sure not to include them in the client end software!!

For details, please see the description in Chapter 1 "File System" of *Senselock EL Developer Guide*.

Sample Introduction

The PC end program first creates a data file whose size is 300 bytes in the root directory using the developer authorization and then reads & writes data files by invoking Senselock EL EXF. The EXF functions written into Senselock EL are as follows:

Input: sub-function number, identifying whether to write data (`CMD_UPDATE = 0x01`) or read data (`CMD_READ = 0x02`). To write data, input destination file ID, offset, length of the data and the data to be written. To read data, input destination file ID, offset and length of data to be read.

Output: If it's to write data, output success flag or error codes. If to read data, output corresponding data upon a successful return or error codes otherwise.

For example, if the following is input:

```
stDataPkgIn.tag = CMD_UPDATE; //Write data
// stDataPkgIn.buff carries the structure data as shown below
stDataBlk.fid = 0xbf20; // objective file ID
stDataBlk.offset = 0;// offset
/* stDataBlk.buff carries the data to be written: "This is the data to be written
   into the objective Senselock EL file." */
```

For the description of how to use TLV structure *stDataPkgIn*, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download

the successfully linked hex file *sample_20_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef20" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Public Header File

The following "sub-function number" and "user-defined" structure are defined in the public header file:

```
/* command group definition */
#define CMD_UPDATE 0x01
#define CMD_READ   0x02

/* user-specific data block */
typedef struct _FILE_OP_BLOCK
{
    unsigned short fid;     // objective file ID
    unsigned short offset; // operating offset
    unsigned char len;     // buffer length(CMD_UPDATE)
                           // read length(CMD_READ)
    unsigned char buff[MAX_BUFF_SIZE];
} FILE_OP_BLOCK;
```

Senselock EL EXF Program

The function below demonstrates how to open a Senselock EL file and write data into it.

```
int update_internal_file(unsigned short fid, unsigned short offset, char
*in, int inlen)
{
    int ret = 0;
    HANDLE hfile = 0;

    ret = _open(fid, &hfile);
    ... // check ret
    ret = _write(hfile, offset, inlen, in);
... // check ret
    ret = _close(hfile);
    ... // check ret
    return ERR_SUCCESS;
}
```

The function below demonstrates how to open a Senselock EL file and read data from it.

```
int  read_internal_file(unsigned  short  fid,  unsigned  short  offset,
                            unsigned short readlen,
                            unsigned char *out, int *outlen)
{
    int ret = 0;
    HANDLE hfile = 0;

    if (*outlen < readlen)
    {
        return ERR_INVALID_PARAMETER;
    }
    ret = _open(fid, &hfile);
    ... // check ret
    ret = _read(hfile, offset, outlen, out);
    ... // check ret
    ret = _close(hfile);
   ... // check ret
    return ERR_SUCCESS;
}
```

Given below is the invocation of the corresponding sub-function codes. Notice that when the input parameter is of multiple bytes, it shall be subject to byte sequence conversion. For example, *offset* is a 2 byte integer and it needs to be converted as shown below.

```
FILE_OP_BLOCK *dblk = (FILE_OP_BLOCK *)input->buff;
LE16_TO_CC(&dblk->fid);
LE16_TO_CC(&dblk->offset);

switch (input->tag)
{
    case CMD_UPDATE:
        length = 0;
         ret = update_internal_file(dblk->fid,
              dblk->offset, dblk->buff, dblk->len);
    break;

    case CMD_READ:
        length = MAX_IO_DATA_SIZE;
         ret = read_internal_file(dblk->fid,
          dblk->offset, dblk->len, output->buff, &length);
    break;

    default:
        error_exit(ERR_INVALID_PARAMETER, NULL, 0);
    break;
}
```

Finally, set the returned data packet *output*. Set the communication buffer and exit. For detailed description of the codes in this part, please see sample16.

Download the successfully linked result *sample_20\hardware\sample_20_hw.hex* to the root directory of Senselock EL and name it “ef20”.

PC End Program

After verifying the developer PIN, invoke `S4WriteFile()` to create a 300-byte data file in Senselock EL dongle. After it is successfully created, verify user PIN, then invoke Senselock EL EXF to write a character string to the data file and finally read this character string out from Senselock EL.

```
dwResult = S4WriteFile(&stS4Ctx, S4_OBJ_FILE_ID, 0, lpBuffer,
dwBuffLen, dwBuffLen, &dwBytesWritten, S4_CREATE_NEW, S4_DATA_FILE);
... // check dwResult

if(!(dwResult = update_internal_file(S4_OBJ_FILE_ID_HEX, 0, lpStr,
strlen(lpStr)+1)))
{
   ...
}
...
if(!(dwResul = read_internal_file(S4_OBJ_FILE_ID_HEX, 0,
(unsigned char)(strlen(lpStr)+1), lpBuffer, &dwBuffLen)))
{
  ...
}
```

The code below encapsulates the input data and invokes Senselock EL to write new contents into the target file:

```
unsigned long update_internal_file(unsigned short fid, unsigned short
offset,
                         char *in, int inlen)
{
    FILE_OP_BLOCK stDataBlk = {0};
    assert(in != NULL);
```

```
    stDataBlk.fid = fid;
    stDataBlk.offset = offset;
    stDataBlk.len = inlen;
    memcpy(stDataBlk.buff, in, inlen);

    stDataPkgIn.tag = CMD_UPDATE;
    stDataPkgIn.len=sizeof(FILE_OP_BLOCK) -
                sizeof(stDataBlk.buff)+ stDataBlk.len; /* Refer
to footnote for comment of this sentence 1*/
    memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
    ... // call S4Execute() and check result

    return 1;
}
```

Questions

Q1. What is the difference between invoking the function for updating data file `update_internal_file()` and invoking directly `S4WriteFile (S4_UPDATE_FILE)` in this sample?

Q2. If no other restrictions are imposed, what are the huge hidden security troubles in this sample?

Q3. What measures shall be taken to eliminate these hidden security troubles?

[1] The *buff* member variable in the user-defined structure is usually based on fixed-length array, but sometimes this array does not carry so many valid data. *Len* indicates the length of valid data it carries.
`sizeof(FILE_OP_BLOCK) - sizeof(stDataBlk.buff)` indicates the size of other data except *buff* array in the user-defined structure. This result plus *stDataBlk.len* is the valid size of the whole user-defined data structure.

Sample 21: How to create a file dynamically in the dongle?

Sample Gist

1. How to invoke SES function `_create()` to create a file dynamically in the Senselock EL dongle?

Senselock EL2.3 introduces a new SES: `_create()`, with which Senselock EL EXF can create a file dynamically in the middle of execution.

2. What precautions shall be taken in the dynamic creation of a file?

- Since the SES function is designed specifically for remote operation, the files created through this function can be accessed only from inside Senselock EL. In other words, even with developer PIN, you cannot invoke `S4WriteFile()` to write the file created with this SES.
- The file created with this function will inherit the security attributes of the parent file (the caller EXF), so to make the newly created file usable, the parent file (the caller EXF) itself shall be set to "readable/writable" in attribute; otherwise the created one will be unusable "dead file".
- The executable file (EXF) created with this function must be enabled by invoking `_enable_exe()` before it can be executed.
- The EXF created by VM executable file is of VM EXF; the EXF created by XA executable file is of XA EXF.

3. Watch out the security issue when operating files. Using this sample directly will cause many hidden security troubles!

For the convenience of description, this sample does not take into account any issue of security. No authorization check is made when the target data file is created. The written data are by no means encrypted. In addition, the data files in the dongle are output directly to the PC end. This will cause great hidden security trouble. In the actual use, to create a file in the dongle shall be subject to authorization check. The input data shall be encrypted (just like what is used on our remote-update platform). Moreover, the use of the content of the read files, unless encrypted, shall be strictly restricted to the dongle.

What is more, this sample contains the "developer" level operation codes. In the release of the software, be sure not to include them in the client end software!!

For details, please see the description in Chapter 1 "File System" of *Senselock EL Developer Guide*.

Sample Introduction

On top of sample 20, this sample demonstrates how to create a file dynamically. The PC end program invokes EXF to create a data file whose size is 300 bytes and then to read & write the data file. The EXF functions written into Senselock EL are as follows:

Input: Add sub-function number (CMD_CREATE=0x03) on top of sample 20, identify to create a file, input parameters including destination file ID, file size, file type and creation flag.

Output: If it is to create a file, output success flag or error codes.

For example, if the following is input:

```
stDataPkgIn.tag = CMD_CREATE; // Create file

// stDataPkgIn.buff carries the following structure data

stDataBlk.fid = 0xbf21;// destination file ID

stDataBlk.fsize = 300; // destination file size

stDataBlk.ftype = SES_FILE_TYPE_EXE_DATA; // the type is data file

stDataBlk.flag = CREATE_OPEN_ALWAYS; // create or open the
destination file
```

For the description of how to use TLV structure *stDataPkgIn* and "user-defined" structure *stDataBlk*, please see sample 16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_21_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef21" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Attention: When downloading the hardware program of the sample using the Development Test Tool of Senselock EL, you need to check the option "R/W security for EXF"; otherwise, by default, this hardware program will be set to "non-readable/non-writable". In this case, the child file it creates dynamically will be a "dead file".

Public Header File

The following "user-defined" structure is added to the public header file:

```
/* user-specific data block */
typedef struct _FILE_CREATE_BLOCK
{
    unsigned short fid;     // objective file ID
    unsigned short fsize;   // file size
    unsigned char  ftype;   // file type
    unsigned char  flag;    // create flag
} FILE_CREATE_BLOCK;
```

Also added are file type definitions quoted from *ses_v3.h* and creation flag (designed primarily for the input of parameters by PC end program):

```
/* create file type, quoted from ses_v3.h */
#define SES_FILE_TYPE_EXE      0x00  /* executable file    */
#define SES_FILE_TYPE_EXE_DATA 0x01  /* data file      */
#define SES_FILE_TYPE_RSA_PUB  0x02/* RSA public key file   */
#define SES_FILE_TYPE_RSA_SEC  0x03/* RSA secret key file */

/* create file flag, quoted from ses_v3.h */
#define CREATE_OPEN_ALWAYS 0x00  /* open if file already exist, create
                                    a new file if not existing    */
#define CREATE_FILE_NEW 0x01   /* create a new file and open it    */
#define CREATE_OPEN_EXISTING 0x02    /* open a file exist  */
```

Senselock EL EXF Program

The function below demonstrates how to create a file dynamically.

```
int   create_internal_file(unsigned   short   fid,   unsigned   short
```

```
fsize,unsigned char bftype,unsigned char bflag)
{
    int ret = 0;
    HANDLE hfile = 0;
    ret = _create(fid, fsize, bftype, bflag, &hfile);
    ... // check ret
    ret = _close(hfile);
   ... // check ret
    return ERR_SUCCESS;
}
```

Other codes are similar to those in sample 20 Should there be any doubt, please see the detailed description of sample 20.

Download the successfully linked result *sample_21\hardware\sample_21_hw.hex* to the root directory of Senselock EL device and name it "ef21". Attention: When downloading the hardware program of the sample using the Development Test Tool of Senselock EL, you need to choose the option "R/W security for EXF".

PC End Program

Similar to the program in sample 20, the difference being that in this sample, the destination file is created by invoking Senselock EL EXF.

```
if(!(dwResult = create_internal_file(S4_OBJ_FILE_ID_HEX,
                  300, SES_FILE_TYPE_EXE_DATA, CREATE_OPEN_ALWAYS)))
{
   ...
}
...
if(!(dwResult  =  update_internal_file(S4_OBJ_FILE_ID_HEX,  0,  lpStr,
                  strlen(lpStr)+1)))
{
   ...
}
...
if(!(dwResul = read_internal_file(S4_OBJ_FILE_ID_HEX, 0,
          (unsigned char)(strlen(lpStr)+1), lpBuffer, &dwBuffLen)))
{
  ...
}
```

The code below encapsulates input data and invokes Senselock EL to create the destination file:

```
int  create_internal_file(unsigned  short  fid,  unsigned  short  fsize,
               unsigned char ftype, unsigned char flag)
{
  FILE_CREATE_BLOCK stDataBlk = {0};
  stDataBlk.fid = fid;
  stDataBlk.fsize = fsize;
  stDataBlk.ftype = ftype;
  stDataBlk.flag = flag;

  stDataPkgIn.tag = CMD_CREATE;
  stDataPkgIn.len = sizeof(FILE_CREATE_BLOCK);
  memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
  ... // call S4Execute() and check result

  return 1;
}
```

Questions

Q1. How to judge the type of a file in the dongle using `_create()` in the running of Senselock EL EXF?

Q2. How to achieve the same goal using Senselock EL API?

5. Mathematical Operations

This chapter introduces how to make single precision and double precision floating point mathematical operations in Senselock EL using SES.

Sample 22: How to use single precision floating point number?

Sample Gist

1. Why do we use Senselock EL SES to operate single precision floats and what are their usages?

   If you need the single precision operation result only, you can choose not to use the hardware SES function as introduced in this sample. You can include *math.h* directly and use the function in it or use the operator directly. However, for VM EXF, using the single precision float SES functions can remarkably enhance operation speed in Senselock EL hardware. For XA EXF, the single precision float function in the hardware will not pick up speed. It is offered for the sake of compatibility.

Sample Introduction

This sample demonstrates how to work out the third side with two sides and an included angle known in a triangle, using the single precision floating point number. The EXF functions written into Senselock EL are as follows:

Input: two sides and their included angle.

Output: the third side.

For example, if the following is input:

```
stDataBlk.angle = 90; // included angle

stDataBlk.side1 = 3.0; // one side

stDataBlk.side2 = 4.0; // another side
```

The third side can be worked out through calculation: 5.0

For the description of how to use "user-defined" structure *stDataBlk*, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_22_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef22" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Public Header File

Added is the following "user-defined" structure:

```
/* user-specific data block */
typedef struct _DATA_BLOCK
{
    float side1;
    float side2;
    float angle;
} DATA_BLOCK;
```

Senselock EL EXF Program

The sentence below converts the input included angle into radian representation:

```
radian = _divf(_multf(dblk->angle,PI), 180);// from angle to radian
```

The function below works out the third side with two sides and one included angle known in a triangle:

```
float calc_tri_side(float side1, float side2, float radian)
{
     return _sqrtf(_subf(_addf(_powf(side1, 2),_powf(side2,
         2)),_multf(2,_multf(_multf(side1,side2),_cosf(radian)))));
}
```

For the description of basic input/output operations and program framework, please see sample16.

Download the successfully linked result *sample_22\hardware\sample_22_hw.hex* to the root directory of Senselock EL device and name it "ef22".

PC End Program

The code below assembles the input data packet and invokes Senselock EL EXF to work out the third side:

```
unsigned long calc_tri_side(float side1, float side2, float angle, float
*p_result)
{
    ...
    stDataBlk.angle = angle;
    stDataBlk.side1 = side1;
    stDataBlk.side2 = side2;

    stDataPkgIn.tag = 0; /* no meaning */
    stDataPkgIn.len = sizeof(DATA_BLOCK);
    memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
   ... // call S4Execute() and check result
    memcpy(p_result, stDataPkgOut.buff, stDataPkgOut.len)
    ...//
}
```

Invoke the above-mentioned function from the main program and print it out:

```
    printf("Calculate the third side...\n");
    if (!calc_tri_side(side1, side2, angle, &result))
    {
    ...
    }
   printf("The result is %2.3f", result);
```

Questions

Sample 23: How to use double precision floating point number?

Sample Gist

1. Particularity in processing double precision floating point numbers

Double precision floating point number must use the SES function in the hardware. As the EXF compiler of the hardware does not support double type (If the key word "`double`" is used in the compiler, it might be processed as "`float`", depending on the setting of the compiler), the double precision floating point number in Senselock EL is represented via a structure (which is defined as `DOUBLE_T`). "=" cannot be used directly to assign value to the double precision floating point.

2. How to assign value to `DOUBLE_T` structure?

It can be done this way:

- By means of memory copying. For example, if we know in advance the representation of the double floating point number by byte in the memory. When this approach is used, notice that DOUBLE_T uses the IEEE real*8 format and always aligns with Little Endian. It is compiler-independent. The double type of Microsoft Visual C++ is also based on the real*8 format as specified by IEEE. On the X86 computer using Little Endian, double can be used directly to assign value to `DOUBLE_T` without the necessity of conversion.
- Some SES functions are used to carry out conversion between single precision floating point number, integer and double precision floating point number, such as `_ftod(…)` etc.
- Use the additionally offered auxiliary function to convert "character string" to "double precision floating point" .

Sample Introduction

This sample demonstrates how to use double precision floating number in the composition of Senselock EL EXF codes. The EXF functions written into Senselock EL are as follows:

Input: operands op1 and op2 of two double types; sub-function number: CMD_DBL_ADD indicates to get summation; CMD_DBL_SUB indicates to get difference; CMD_DBL_MLT indicates to get product; CMD_DBL_DIV indicates to get quotient.

Output: calculation result of corresponding double type.

For example, if the following is input:

```
stDataBlk.cmd = CMD_DBL_ADD; // for summation
stDataBlk.op1 = 1.7976e+50;
stDataBlk.op2 = 3.4028e+50;
```

The result worked out through calculation is 5.2004e+50

For the description of how to use the "user-defined" structure *stDataBlk*, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_23_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef23" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Public Header File

The following "user-defined" structure is defined:

```
/* user-specific data block */
typedef struct _DATA_BLOCK
{
    unsigned char cmd;
   DOUBLE_T op1; // operand 1
   DOUBLE_T op2; // operand 2
} DATA_BLOCK;
```

Where *cmd* stands for sub-function number whose possible value is

```
/* command group definition */
#define CMD_DBL_ADD 0x00   // double add
#define CMD_DBL_SUB 0x01   // double sub
#define CMD_DBL_MLT 0x02   // double multiply
#define CMD_DBL_DIV 0x03   // double divide
```

Senselock EL EXF Program

Attention: Although it is multi-byte double data transported from the PC end, `DOUBLE_T` is based on the Little Endian alignment mode, so there is no need for byte sequence conversion.

The algorithm is quite simple. Given below is the code that does calculation based on the sub-function number:

```
    switch (dblk->cmd)
    {
        case CMD_DBL_ADD:
            ret = _add(&result, &dblk->op1, &dblk->op2);
        break;
        case CMD_DBL_SUB:
            ret = _sub(&result, &dblk->op1, &dblk->op2);
        break;
        case CMD_DBL_MLT:
            ret = _mult(&result, &dblk->op1, &dblk->op2);
        break;
        case CMD_DBL_DIV:
            ret = _div(&result, &dblk->op1, &dblk->op2);
        break;
        default:
            error_exit(ERR_INVALID_PARAMETER, NULL ,0);
        break;
    }
```

For the description of basic input/output operations, please see sample16.

Download the successfully linked result *sample_23\hardware\sample_23_hw.hex* to the root directory of Senselock EL device and name it "ef23".

PC End Program

DOUBLE_T type is identical with double type in memory storage under the VC compiler environment of X86:

```
#ifdef WIN32
#define  DOUBLE_T double
#endif
```

Assemble the input data packet and invoke the code of Senselock EL EXF:

```
stDataBlk.cmd = CMD_DBL_ADD;
stDataBlk.op1 = 1.7976e+50;
stDataBlk.op2 = 3.4028e+50;
stDataPkgIn.tag = 0; /* no meaning */
stDataPkgIn.len = sizeof(DATA_BLOCK);
memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
... // Call S4Execute() and check result
```

If it succeeds, process the output data packet.

```
printf("The result:%e\n", *((double *)stDataPkgOut.buff));
```

Questions

Q1. If double d1 = 2 and float f1 = 2 are defined in VC, what are their IEEE representation? How are they stored in the PC end memory? After they are transported into Senselock EL, can they assign value directly to the DOUBLE_T structure variable or float variable?

Sample 24: How to avoid double precision floating point number boundary inaccuracy?

Sample Gist

1. Why is there double precision floating point number boundary inaccuracy?

Senselock EL uses real*8 format as specified by IEEE for double precision floating point number. This format itself is an approximate representation of precision data. Moreover, due to the representation range of double precision and the issue of border correction by the double precision library, there might occur a great inaccuracy in operation under some border conditions.

2. How to avoid double precision floating point number boundary inaccuracy?

If the data involved in computation fall within the border range susceptible to inaccuracy, measures shall be taken to avoid errors. For the details on inaccuracy border conditions, please refer to the description of Chapter 8 "Double Precision Floating Point Number Boarder Conditions" of *Senselock EL Developer Guide*.

Sample Introduction

This sample demonstrates how to work out the trigonometric function value of a DOUBLE_T number with Senselock EL, shows an example of inaccuracy-induced calculation error and finally presents a correct way to prevent the error from occurring. The EXF functions written into Senselock EL are as follows:

Input: sub-function number, identifying the desired result of sine (CMD_DBL_SIN), cosine (CMD_DBL_COS), tangent (CMD_DBL_TAN), arcsine (CMD_DBL_ASIN), arc cosine (CMD_DBL_ACOS), arc tangent (CMD_DBL_ATAN) and a double precision operand.

Output: corresponding trigonometric function operation result.

For example, if the following is input:

```
stDataBlk.cmd = CMD_DBL_COS; // to get cosine result
stDataBlk.operand = 0;
```

The calculated result is 1.0000000000

For the description of how to use "user-defined" structure *stDataBlk*, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_24_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef24" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Public Header File

The following "user-defined" structure is defined:

```
/* user-specific data block */
typedef struct _DATA_BLOCK
```

```
{
    unsigned char cmd;
   DOUBLE_T operand;
} DATA_BLOCK;
```

Where *cmd* stands for sub-function number whose possible value is:

```
/* command group definition */
#define CMD_DBL_SIN 0x00 // double sin
#define CMD_DBL_COS 0x01 // double cos
#define CMD_DBL_TAN 0x02 // double tan
#define CMD_DBL_ASIN 0x03 // double asin
#define CMD_DBL_ACOS 0x04 // double acos
#define CMD_DBL_ATAN 0x05 // double atan
```

Senselock EL EXF Program

The algorithm is quite simple. Given below is the code that does calculation based on the sub-function number:

```
    DATA_BLOCK *dblk = (DATA_BLOCK *)input->buff;
    switch (dblk->cmd)
    {
        case CMD_DBL_SIN:
            ret = _sin(&result, &dblk->operand);
        break;
        case CMD_DBL_COS:
            ret = _cos(&result, &dblk->operand);
        break;
        case CMD_DBL_TAN:
            ret = _tan(&result, &dblk->operand);
        break;
        case CMD_DBL_ASIN:
            ret = _asin(&result, &dblk->operand);
        break;
        case CMD_DBL_ACOS:
            ret = _acos(&result, &dblk->operand);
        break;
        case CMD_DBL_ATAN:
            ret = _atan(&result, &dblk->operand);
        break;
        default:
            error_exit(ERR_INVALID_PARAMETER, NULL, 0);
        break;
    }
```

For the description of basic input/output operations, please see sample16.

Download the successfully linked result *sample_24\hardware\sample_24_hw.hex* to the root directory of Senselock EL device and name it “ef24”.

PC End Program

Assemble the input data packet and work out the cosine value of the *operand*. In this sample, the *operand* value is 0:

```
    stDataBlk.cmd = CMD_DBL_COS;
    stDataBlk.operand = operand;
    stDataPkgIn.tag = 0; /* no meaning */
    stDataPkgIn.len = sizeof(DATA_BLOCK);
    memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
   ...// Call S4Execute and check result
```

If it succeeds, process the output data packet.

```
printf("The result:_cos(%.10f) = %.10f\n", operand,
        *((double *)stDataPkgOut.buff));
```

The program will output the following at this point of execution:

"The result:_cos(0.0000000000) = 1.0000000000"

Send back the returned value and work out the arc cosine value:

```
/* if no adjustment, there will be unexpected wrong result */
    stDataBlk.cmd = CMD_DBL_ACOS;
    memcpy(&stDataBlk.operand, stDataPkgOut.buff,
          sizeof(double));
    stDataPkgIn.tag = 0; /* no meaning */
    stDataPkgIn.len = sizeof(DATA_BLOCK);
    memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
    ...// Call S4Execute and check result
```

After it is correctly executed, the program outputs unexpected error result:

"Unexpected wrong result: _acos(_cos(0.0000000000)) = 1.5707963268"

Why is that? Check Chapter 8 "Double Precision Floating Point Number Border Conditions" of *Senselock EL Developer Guide* and we will know: in terms of SES `_cos(…)`, "when the operand is close to the integer multiple of π/2, there will be a great inaccuracy."

Given below is the code for cosine value fine tuning based on the operand range:

```
/* adjustment, first, make bias in [0,2PI] */
bias  = fabs(operand) - (int)(fabs(operand) / (2*PI))*2*PI;
if ( bias < DBL_EPSILON)
{
    cos_result = 1;
}
else if (fabs(bias - PI/2) < DBL_EPSILON)
{
    cos_result = 0;
}
else if (fabs(bias - PI) < DBL_EPSILON)
{
    cos_result = -1;
}
else if (fabs(bias - PI*3/2) < DBL_EPSILON)
{
    cos_result = 0;
}
```

Input once again the fine-tuned value and work out the arc cosine value:

```
    stDataBlk.cmd = CMD_DBL_ACOS;
    stDataBlk.operand = cos_result;
    stDataPkgIn.tag = 0; /* no meaning */
    stDataPkgIn.len = sizeof(DATA_BLOCK);
    memcpy(stDataPkgIn.buff, &stDataBlk, stDataPkgIn.len);
    ...//Call S4Execute() and check result
```

A correct result is output this time:

"The correct result after adjustment:_acos(_cos(0.0000000000)):0.0000000000"

Questions

Q1. How to write different fine-tuning codes for other trigonometric functions?

6. Cryptographic Algorithms

This chapter introduces how to use the cryptographic algorithms built in Senselock EL as well as how to write programs to expand these algorithms.

The internal SES of Senselock EL offers four categories of basic cryptographic algorithms: asymmetric cryptographic algorithm (RSA), symmetric cryptographic algorithm (DES/TDES), Hash algorithm (SHA1) and real random number generator. These cryptographic algorithms can not only protect the copyright of software effectively, but more importantly extend software towards network involvement trend, such as achieving remote updating in a real sense.

To use these cryptographic algorithms correctly, you need to achieve some basic understandings of the cryptography algorithms and applications. Instead of a specialized learning course for cryptography, this text makes some necessary descriptions of the algorithms and protocols involved. To achieve a deeper understanding, you need to consult related specialized books.

Sample 25: How to encrypt data using DES/TDES?

Sample Gist

1. About DES/TDES and Operation Mode

DES/TDES encryption is widely used to the protection of a huge amount of data. Compared with RSA encryption algorithm, they are much faster. Different operation modes can be selected when DES/TDES are used, such as Electronic Code Book (ECB), Cipher Block Chaining (CBC) and Cipher-FeedBack. Here we are going to demonstrate the usage of two modes: ECB and CBC.

The SES functions `_des_xxx()` and `_tdes_xxx()` of Senselock EL are both of ECB mode. To use CBC mode requires encapsulation on top of this.

The security of DES/TDES is based on secret key. When EXF in the sample is used directly, make sure to change the original secret key built in EXF.

Sample Introduction

This sample demonstrates how to encrypt and decrypt data through DES/TDES under ECB and CBC modes using Senselock EL. The required secret key is already built in EXF, whose value is "0x0001020304050607"(DES) and: "0x000102030405060708090a0b0c0d0e0f"(TDES).

The EXF functions written into Senselock EL EXF are as follows:

Input: sub-function number, respectively can be: `DES_ENC_ECB`（DES ECB mode encryption）, `DES_DEC_ECB`（DES ECB mode decryption）, `TDES_ENC_ECB`（TDES ECB mode encryption）, `TDES_DEC_ECB`（TDES ECB mode decryption）, `DES_ENC_CBC`（DES CBC mode encryption）, `DES_DEC_CBC`（DES CBC mode decryption）, `TDES_ENC_CBC`（TDES CBC mode encryption）, `TDES_DEC_CBC`（TDES CBC mode decryption）followed by the data to be processed and data length. If it is CBC mode, an 8-byte initial vector value shall also be input.

Output: encryption/decryption result.

For example, if the following is input:

```
stDataPkgIn.tag = DES_ENC_ECB; // DES ECB Mode encryption
stDataPkgIn.len = 16; // plain text data length;
// stDataPkgIn.buff carries the following plain text data:
[00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff:]
```

The encryption result is:

```
[3e:f0:a8:91:cf:8e:d9:90:86:52:cf:6b:1c:7b:ab:b4:]
```

For the description of how to use TLV structure *stDataPkgIn*, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_25_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef25" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Public Header File

The following sub-function number definition is added to the public header file:

```
/* command group definition */
#define DES_ENC_ECB 0x00
#define DES_DEC_ECB 0x01
#define DES_ENC_CBC 0x02
#define DES_DEC_CBC 0x03
#define TDES_ENC_ECB 0x04
#define TDES_DEC_ECB 0x05
#define TDES_ENC_CBC 0x06
#define TDES_DEC_CBC 0x07
```

In addition, the following "user-defined" structure (for the convenience of data input by users under CBC mode) is also defined:

```
/* user-specific data block for CBC mode */
typedef struct _DATA_BLOCK
{
    unsigned char iv[8]; // initial vector
    unsigned char len;
    unsigned char buff[MAX_DATA_BLOCK_LEN];
} DATA_BLOCK;
```

Senselock EL EXF Program

The function below demonstrates how to do encryption and decryption under DES/TDES ECB mode.

```
int enc_dec_ecb(unsigned char cmd, unsigned char *key, unsigned char *in,
      unsigned char inlen, unsigned char *out, unsigned char *outlen)
{
    ...
    switch(cmd)
    {
        case DES_ENC_ECB:
            ret = _des_enc(key, inlen, tmp_buf);
```

```
        break;
        case DES_DEC_ECB:
            ret = _des_dec(key, inlen, tmp_buf);
        break;
        case TDES_ENC_ECB:
            ret = _tdes_enc(key, inlen, tmp_buf);
        break;
        case TDES_DEC_ECB:
            ret = _tdes_dec(key, inlen, tmp_buf);
        break;
        default:
            return ERR_INVALID_PARAMETER;
    }
    ...
}
```

The function below demonstrates how to do encryption and decryption under DES/TDES CBC mode. It differs mainly from ECB in the use of (initial vector, iv):

```
int enc_dec_cbc(unsigned char cmd, unsigned char *key, unsigned
                char *iv, unsigned char *in,
                unsigned char inlen,
                unsigned char *out, unsigned char *outlen)
{
   ret = _mem_copy(tmp_iv, iv, sizeof(tmp_iv));
    ...//
    for (i = 0; i < length/8; i++)
    {
        /* Chain if encrypting.
         */
        if (cmd == DES_ENC_CBC || cmd == TDES_ENC_CBC)
        {
            for (j = 0; j < 8; j++)
            {
                tmp_buf[i*8+j] ^= tmp_iv[j];
            }
            if (cmd == DES_ENC_CBC)
            {
                ret = _des_enc(key, 8, &tmp_buf[i*8]);
            }
            else
            {
                ret = _tdes_enc(key, 8, &tmp_buf[i*8]);
            }
            ret = _mem_copy(tmp_iv, &tmp_buf[i*8], 8);
```

```
        }
        else
        {
            ret = _mem_copy(inputBlock, &tmp_buf[i*8], 8);
            if (cmd == DES_DEC_CBC)
            {
                ret = _des_dec(key, 8, &tmp_buf[i*8]);
            }
            else
            {
                ret = _tdes_dec(key, 8, &tmp_buf[i*8]);
            }
            for (j = 0; j < 8; j++)
            {
                tmp_buf[i*8+j] ^= tmp_iv[j];
            }
            ret = _mem_copy(tmp_iv, inputBlock, 8);
        }

    }
    ...//
}
```

Given below is the code for pointer conversion and sub-function invocation. Notice here DES uses the former 8 bytes of the global array “key” as secret key while TDES uses all the 16 bytes of the array as secret key.

```
    switch(input->tag)
    {
        case DES_ENC_ECB:
        case DES_DEC_ECB:
        case TDES_ENC_ECB:
        case TDES_DEC_ECB:
            length = sizeof(tmp_buf);
            ret = enc_dec_ecb(input->tag, key, input->buff,
                 input->len, &output->buff, &length);
        break;
        case DES_ENC_CBC:
        case DES_DEC_CBC:
        case TDES_ENC_CBC:
        case TDES_DEC_CBC:
            dblk = (DATA_BLOCK *)input->buff;
            length = sizeof(tmp_buf);
            ret = enc_dec_cbc(input->tag, key, dblk->iv,
                 dblk->buff, dblk->len, &output->buff, &length);
```

```
        break;
        default:
            error_exit(ERR_INVALID_PARAMETER, NULL, 0);
    }
```

Finally, set the returned data packet *output* and communication buffer and exit. For detailed description of the codes in this part, please see sample16.

Download the successfully linked result *sample_25\hardware\sample_25_hw.hex* to the root directory of Senselock EL device and name it "ef25".

PC End Program

The main program invokes one by one all the sub-function numbers of Senselock EL EXF, performs a series of encryption/decryption operations under DES /TDES ECB and CBC mode and outputs the result. For the convenience of use, we have encapsulated the code of each sub-function to be invoked. Given below is the list of invocation sequence of encapsulated functions:

```
  ...
if (!(dwResult = enc_dec_ecb(DES_ENC_ECB, plainText, plainTextLen,
cipher, &cipherLen)))/* DES ECB Encryption */
  {
  ...
  }
  ...
if (!(dwResult = enc_dec_ecb(DES_DEC_ECB, cipher, cipherLen,
plainText, &plainTextLen))) /* DES ECB Decryption */
  {
  ...
  }
  ...
if (!(dwResult = enc_dec_ecb(TDES_ENC_ECB, plainText, plainTextLen,
cipher, &cipherLen))) /* TDES ECB Encryption */
  {
  ...
  }
if (!(dwResult = enc_dec_ecb(TDES_DEC_ECB, cipher, cipherLen,
plainText, &plainTextLen))) /* DES ECB Decryption */
  {
  ...
  }
  ...
if (!(dwResult = enc_dec_cbc(DES_ENC_CBC, iv, plainText,
plainTextLen, cipher, &cipherLen))) /* DES CBC Encryption */
  {
  ...
```

```
    }
    ...
if (!(dwResult = enc_dec_cbc(DES_DEC_CBC, iv, cipher, cipherLen,
plainText, &plainTextLen))) /* DES CBC Encryption */
    {
    ...
    }
    ...
if (!(dwResult = enc_dec_cbc(TDES_ENC_CBC, iv, plainText,
plainTextLen, cipher, &cipherLen))) /* TDES CBC Encryption */
    {
    ...
    }
    ...
if (!(dwResult = enc_dec_cbc(TDES_DEC_CBC, iv, cipher, cipherLen,
plainText, &plainTextLen))) /* TDES CBC Encryption */
    {
    ...
    }
```

For details on how to encapsulate the input data and basic function framework, please see sample16.

Questions

Q1. In the sample, what are the maximum bytes of data to be encrypted that the encapsulated function: `enc_dec_ecb()` and `enc_dec_cbc()` at the PC end can process at once?

Q2. How to modify them so that they can process more data to be encrypted?

Q3. When the length of the input data is not multiple of 8, how does the function `enc_dec_cbc（）` in EXF program of this sample do padding? Are there any other substitute padding schemes? What are the advantages and disadvantages of each padding scheme?

Sample 26: How to use RSA for data encryption/decryption, digital signature and verification?

Sample Gist

1. About RSA Algorithm and PKCS#1 Encoding

The RSA algorithm, the first relatively flawless public key algorithm, can be used for encryption and digital signature. Of the public key algorithms available, RSA is the easiest to understand and implement. RSA is named after its three inventors Ron Rivest, Adi Shamir and Leonard Adleman. This algorithm has stood years of indepth cryptographic analysis. Although cryptographic analysts can neither prove nor deny the security of RSA, this exactly shows there is some credibility in the algorithm. PKCS（Public Key Cryptographic Standard）is a series of specifications governing the public key algorithm issued by RSA lab. When signing a Hash value and encrypting the plain text, we can use PKCS#1 for encoding.

2. Some limitations of using current SES for signing/signature verification

`_rsa_enc()` and `_rsa_dec()` are two basic RSA operation SES(s). You can fulfill all RSA functions like encryption/decryption, signing and verification using the two SES(s). To make easy signing and verification operations under some circumstances, Senselock EL offers two additional SES functions: `_rsa_sign()` and `_rsa_veri()` for signing and verification, but these two functions can not meet all the needs for digital signing and verification. For example,

- `_rsa_sign()` and `_rsa_veri()` support only one kind of Hash algorithm：SHA1;
- When `_rsa_veri()` verifies signature, the Hash value of plain text shall be acquired through operation of the SES functions `_sha1_xxx`;

If the existing SES function cannot meet your needs for signing and verification, you can create your own signing and verification function based on `rsa_enc()` and `_rsa_dec()`. This sample demonstrates how this approach is actually implemented.

Sample Introduction

This sample demonstrates how to use Senselock EL for data encryption/decryption, digital signing and verification.

The EXF functions written into Senselock EL are as follows:

Sub-function number	Input	Output
CMD_SIGN （RSA Sign）	Private key ID, Hash algorithm (supporting currently only MD5 and SHA1, defined respectively as DA_MD5 and DA_SHS),	Signature

	Hash value and length.	
CMD_VERIFY (RSA Verification)	Public key file ID, Hash algorithm (ditto), Hash value and length, signature to be verified and its length.	Success or failure
CMD_ENCRYPT （RSA Encrypt）	Public key file ID, plain text and length	Ciphertext
CMD_DECRYPT （RSA Decrypt）	Private key file ID, ciphertext and length	Plain text

For example, if the following is input:

`stDataPkgIn.tag = CMD_SIGN;` // digital signature. The data part carries the following structure:

```
stRsaBlk.fid = 0xef02; // private key file ID
stRsaBlk.dgst_alg = DA_MD5;
stRsaBlk.len = 0x10; // Fixed-length of any MD5 hash value: 16
memcpy(stRsaBlk.buff,in,inlen); // "in" contains 16-byte Hash value
```

Output: 128-byte signature

For the description of how to use TLV structure *stDataPkgIn* and user-defined structure, please see sample16.

Sample Implementation

First generate a RSA public/private key pair whose IDs are respectively "ef01" and "ef02" in the dongle using Senselock EL Development Test Tool and then compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_26_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef26" and then run PC End Program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Public Header File

The following sub-function number definitions are added to the public header file:

```
/* command group definition */
#define CMD_SIGN    0x01
#define CMD_VERIFY  0x02
#define CMD_ENCRYPT 0x03
#define CMD_DECRYPT 0x04
```

In addition, the following “user-defined” structure (for the convenience of multiple user data input at the time of signing/verification) is defined:

```
/* user-specific data block */
typedef struct _RSA_BLOCK
{
    unsigned short fid; // File ID of the private/public key.
    unsigned char dgst_alg; // digest algorithm
    unsigned char len;
    unsigned char buff[RSA_BLOCK_LEN];
} RSA_BLOCK;

typedef struct _RSA_VERIFY_BLOCK
{
    unsigned short fid; // File ID of the public key.
    unsigned char dgst_alg;
    unsigned char dgst[MAX_HASH_BLOCK_LEN]; // digest
    unsigned char dgst_len;
    unsigned char sig[RSA_BLOCK_LEN]; // signature
    unsigned char sig_len;
} RSA_VERIFY_BLOCK;
```

Senselock EL EXF Program

The function below demonstrates how to make RSA signature with PKCS#1 encoding. There are many limitations in the use of the SESs `_rsa_sign()` and `_rsa_veri()` built in Senselock EL. The codes in this sample eliminate these limitations, so it is recommended to use them instead for RSA signing/signature verification. The part as listed below is designed primarily to do data padding under PKCS#1 encoding mechanism.

```
int make_signature(unsigned short fid, unsigned char dgst_alg,
                           unsigned char *in, int inlen,
                           unsigned char *out, int *outlen)
{
    ...
    /* PKCS#1 padding */
    tmp_buf[0] = 0;
    /* block type 1 */
    tmp_buf[1] = 1;
    /* padding */
    for (i=2; i<RSA_BLOCK_LEN-1-dgst_infolen-r_blklen; i++)
    {
        tmp_buf[i]=0xff;
    }
    /* seperator */
    tmp_buf[i++] = 0;
```

```
    ret = _mem_copy(&tmp_buf[i], p_dgst_info, dgst_infolen);
    // check ret
    i += dgst_infolen;
    ...
    ret = _rsa_dec(RSA_CALC_NORMAL, fid, RSA_BLOCK_LEN, tmp_buf);
    ...
}
```

`verify_signature()` will conduct check based on the format of the PKCS#1 encoding. For the sake of space, no detailed description will be given here.

It is quite simple to implement RSA encryption/decryption functions `rsa_encrypt()` and `rsa_decrypt()` under PKCS#1 encoding: simply invoke the SES functions of Senselock EL: `_rsa_enc()` and `_rsa_dec()`. For details, please see source codes.

Conduct pointer conversion in the main program, make byte sequence conversion for the input multi-byte parameters and invoke the corresponding sub-function. Finally, set returned data packet *output* as well as communication buffer and exit. For the detailed description of the program framework, please see sample16.

Download the successfully linked result *sample_26\hardware\sample_26_hw.hex* to the root directory of Senselock EL device and name it "ef26".

PC End Program

The main program invokes one by one the sub-function numbers of Senselock EL EXF, accomplishes signing/signature verification and encryption/decryption operations under RSA PKCS#1 encoding and output them. Since the signature function `make_signature()` accepts only Hash value input, we first make the operation of working out at the PC end the Hash value for the data to be signed using the software algorithm library offered by SDK. As for how to use the software algorithm library *Cryptst.lib* at the PC end, please see Appendix A "Introducing Cryptographic Algorithm Usage" of *Senselock EL Developer Guide*.

```
dwResult = Digest(DA_MD5,lpToBeSigned,sizeof(lpToBeSigned),
              lpDigest,&dwDigestLen);
...// check dwResult
```

For the convenience of use, we have encapsulated the code of each sub-function to be invoked.. Given below is a list of invocation sequence for the encapsulated functions:

```
if (!(dwResult = make_signature(0xef02,DA_MD5,lpDigest,
                      dwDigestLen,lpSignature,&dwSignatureLen)))
{
...
}
if (!(dwResult = verify_signature(0xef01,DA_MD5,lpDigest
                  ,dwDigestLen,lpSignature,dwSignatureLen)))
{
```

```
...
}
if (!(dwResult = rsa_encrypt(0xef01,lpPlainText,
            sizeof(lpPlainText),lpCipherText,&dwCipherTextLen)))
{
...
}
if (!(dwResult = rsa_decrypt(0xef02,lpCipherText,
              dwCipherTextLen,lpPlainText,&dwPlainTextLen)))
{
...
}
```

For the ways of encapsulating the input data and the basic function framework, please see sample16.

Questions

Q1. If SES functions `_rsa_sign()` and `_rsa_veri()` must be used directly for RSA data signing/signature verification, how to achieve this? Are there any inconvenience?

Q2. How to modify the functions `make_signature()` and `verify_signature()` in EXF so that they can support more Hash algorithms?

Sample 27: How to work out H-MAC Message Authentication Code?

Sample Gist

1. About Message Authentication Code

MAC (Message Authentication Code) is usually designed to check the integrity of data. MAC created through Hash algorithm is usually called HMAC. It is called HMAC-SHA1 or HMAC-MD5 depending on the Hash function selected. This sample demonstrates the way HMAC-SHA1 is implemented. The way HMAC is implemented via MD5 is more or less the same.

Besides Hash algorithms, you can also calculate the MAC using algorithms like DES. We will introduce the way DES-MAC is implemented in sample 28.

The security of H-MAC is based on the secret key. When EXF in this sample is used directly, make sure to change the original secret key built in EXF.

Sample Introduction

This sample demonstrates how to work out the message authentication code of an input message with HMAC-SHA1 algorithm in Senselock EL. The secret key built in EXF is "0x0102030405060708090a0b0c0d0e0f10111213141516171819".

The EXF functions written into Senselock EL are as follows:

Input: message and length

Output: HMAC-SHA1 MAC value

For example, input: the message with 50-byte "0xcd" as follows:

```
stDataPkgIn.len = 50;

memset(stDataPkgIn.buff, 0xcd, 50);
```

Output: HMAC-SHA1 value is:

```
[4c:90:07:f4:02:62:50:c6:bc:84:14:f9:bf:50:c8:6c:2d:72:35:da:]
```

For the description of how to use TLV structure *stDataPkgIn* and "user-defined" structure, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_27_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef27" and then run PC End Program. As for how

to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Senselock EL EXF Program

The function below completes the algorithm for working out HMAC-SHA1:

```
int hmac_sha(char*    k,     /* secret key */
             int      lk,    /* length of the key in bytes */
             char*    d,     /* data */
             int      ld,    /* length of data in bytes */
             char*    out,   /* output buffer, at least "t" bytes */
             int      *t)
{
   ...// Call SES functions _sha1_init (), _sha1_update(),
                          sha1_final() for several times
}
```

Invoke the sub-function in the main program and then set returned data packet *output* as well as communication buffer and exit. For detailed description of the program framework, please see sample16.

Download the successfully linked result *sample_27\hardware\sample_27_hw.hex* to the root directory of Senselock EL device and name it “ef27”.

PC End Program

The main program encapsulates the function designed to work out HMAC-SHA1

```
/* Function to compute the digest */
int hmac_sha(char*    in,     /* data */
             int      inlen,    /* length of data in bytes */
             char*    out,   /* output buffer, at least "t" bytes */
             int      *outlen)
{
   ...// Operation for invoking Senselock EL dongle EXF
}
```

For the ways of encapsulating the input data and the basic function framework, please see sample16.

Questions

Q1. How to achieve HMAC-MD5 with reference to this sample and sample 29?

Sample 28: How to work out DES-MAC code?

Sample Gist

1. About DES-MAC code

Set the initial vector iv to all zero and then encrypt the input message with secret key under DES-CBC mode. The last 8-digit cryptogram is DES-MAC code. For MAC, please see sample 28. For encryption/decryption under CBC mode, please see sample26.

The security of DES-MAC is based on secret key. When EXF in this sample is used directly, make sure to change the original secret key built in EXF.

Sample Introduction

This sample demonstrates how to work out the MAC of an input message through DES-MAC in Senselock EL. The secret key value built in EXF is "0x000102030405060708090a0b0c0d0e0f".

The EXF functions written into Senselock EL are as follows:

Input: message and length

Output: CBC-MAC code

For example, if the following is input:

stDataPkgIn.len = 16;

The data of stDataPkgIn.buff is "Original Message"

Output: CBC-MAC code: [6f:3c:99:ea:84:8d:e1:f2:]

For the description of how to use TLV structure stDataPkgIn and user-defined structure, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_28_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef28" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Senselock EL EXF Program

The function below demonstrates how to work out CBC-MAC value. The way the algorithm is implemented is similar to the operation of DES/TDES under CBC mode (except that the initial vector iv is set to all zero). No details will be given here.

```
int cbc_mac(unsigned char cmd, unsigned char *key, unsigned char *in,
                            unsigned char inlen,
                  unsigned char *out, unsigned char *outlen)
{
    ...
    ret = _mem_set(tmp_iv, 0, sizeof(tmp_iv)); // zerorize iv
    ...
}
```

Invoke the sub-function in the main program and then set returned data packet *output* as well as communication buffer and exit. For detailed description of the program framework, please see sample16.

Download the successfully linked result *sample_28\hardware\sample_28_hw.hex* to the root directory of Senselock EL device and name it "ef28".

PC End Program

The main program encapsulates the function designed to work out CBC-MAC

```
int cbc_mac(unsigned char cmd, unsigned char *in,  unsigned
       char inlen, unsigned char *out, unsigned char *outlen)
{
   ...// Operations for invoking Senselock EL dongle
}
```

For the ways of encapsulating the input data and the basic function framework, please see sample16.

Questions

Q1. How to make the functions in sample 25 into a hardware function library so that this sample can use its CBC encryption functions directly?

Sample 29: How to implement MD5 algorithm in Senselock EL?

Sample Gist

1. About MD5 Algorithm

The information digest algorithm MD5 is extensively used in different fields. However, the latest research shows there might be some hidden flaw in this algorithm. Although no fatal weakness has been found so far, it is not recommended to use it any more. Thus, this sample aims merely to introducing to users how to extend the self-defined algorithms in Senselock EL.

Sample Introduction

This sample demonstrates how to implement MD5 algorithm with Senselock EL. In addition, this sample writes MD5 algorithm into a hardware function library in the way as introduced in sample 19. Senselock EL EXF project contains this function library and links together to generate the destination EXF.

The EXF functions written into Senselock EL are as follows:

Sub-function number	Input data	Output
CMD_MD5_INIT (initialize algorithm)	None	None
CMD_MD5_UPDATE (Hash data)	Data and length	None
CMD_MD5_FINAL (get Hash value)	None	16-byte MD5 Hash value

```
For example, firstly, initialize the operation:
    stDataPkgIn.tag = CMD_MD5_INIT;
    stDataPkgIn.len = 0;
    // Secondly, inputs data:
    stDataPkgIn.tag = CMD_MD5_UPDATE;
    stDataPkgIn.len = 3;
    // stDataPkgIn.buff contains "abc"
    // Thirdly, inputs data again:
    stDataPkgIn.tag = CMD_MD5_UPDATE;
    stDataPkgIn.len = 4;
```

```
    // stDataPkgIn.buff contains "defg"

    Finally, get the hash value:

    stDataPkgIn.tag = CMD_MD5_FINAL;

    stDataPkgIn.len = 0;

Output: 16 byte hash value:
[7a:c6:6c:0f:14:8d:e9:51:9b:8b:d2:64:31:2c:4d:64:]
```

For the description of how to use TLV structure *stDataPkgIn* and user-defined structure, please see sample16.

Sample Implementation

First compile and link *sample_29_hw_md5_lib.Uv2* project into *sample_29_hw_md5_lib.LIB* and then compile and link the Senselock EL EXF program as mentioned below together with *sample_29_hw_md5_lib.LIB* using Keil. Download the successfully linked hex file *sample_29_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef29" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Public Header File

The following sub-function number definitions are added to the public header file:

```
/* command group definition */
#define CMD_MD5_INIT 0x00
#define CMD_MD5_UPDATE 0x01
#define CMD_MD5_FINAL 0x02
```

Senselock EL EXF Program

The implementation codes in the hardware function library *sample_29_hw_md5_lib.LIB* are relatively quite complicated as they are embedded with much expertise in the area of "code memory", shared memory, byte sequence conversion, etc. No details will be given here.

It shall be noted that when this sample is used to make multiple operations, the temporary variable *md5Ctx* of `MD5_HASH_CONTEXT` type is employed as shared memory variable. It shall be placed at the high end of the memory so that it will not be over-written by other EXF data:

```
/* defined in high xdata area to avoid being overlapped */
DEFINE_AT(MD5_HASH_CONTEXT , md5Ctx, 0x400, RAM_EXT);
```

Carry out pointer conversion and byte sequence conversion for the input byte parameter and invoke the corresponding sub-function. Finally set the returned data packet *output* as well as communication buffer and exit. For the detailed description of the program framework, please see sample16.

Download the successfully linked result *sample_29\hardware\sample_29_hw.hex* to the root directory of Senselock EL device and name it "ef29".

PC End Program

The main program invokes one by one all the sub-function numbers of Senselock EL EXF and performs an operation sequence designed to work out MD5 Hash value. For the way of encapsulating the input data and the basic function framework, please see sample16.

Questions

Sample 30: How to use random numbers in simple communication encryption?

Sample Gist

1. Why do we need communication encryption?

Sometimes, the input/output of the algorithm transplanted into Senselock EL EXF might be quite simple and thus its law can be revealed or get exhausted easily. In this case, we can encrypt communication so that each output will be different even when the processing result is of the same.

2. Some Precautions:

- This sample shows the simplest encryption of communication. You can add some changes and design some more complex methods, but make sure to hide your own encryption thoughts from crackers.
- When the encrypted communication data are decrypted at the PC end, you shall also exercise caution lest they should be tracked down easily by the cracker.

Sample Introduction

This sample demonstrates how to encrypt communication with random numbers when composing Senselock EL programs for data input/output processing. For the convenience of description, we use the MD5 algorithm as introduced in sample 29, yet the operation of working out Hash value will be done in one dongle invocation. Furthermore, same data input will output different values. This is because we make exclusive-OR operation on the Hash value using the 1-byte random number generated in the dongle and output the result following the exclusive-OR operation together with the random number. On receiving the result, the PC end needs only to make exclusive-OR operation using this random number again before restoring the real Hash value.

The EXF functions written into Senselock EL are as follows:

Input: data and length

Output: encrypted MD5 Hash value

For example, if the following is input:

stDataPkgIn.len = 3; // 3-byte data

The data in stDataPkgIn.buff are character string “abc”

Output: 17-byte data

```
[0c:9d:cc:04:a0:4e:d3:2c:4a:0a:a3:e1:b4:7d:e3:ee:9c:];
```

The Hash value decrypted by the PC end is:

```
[90:01:50:98:3c:d2:4f:b0:d6:96:3f:7d:28:e1:7f:72:]
```

Input for the second time the same data:

```
stDataPkgIn.len = 3; // 3-byte data
```

The data in stDataPkgIn.buff are character string "abc"

Output: different values

```
[3f:ae:ff:37:93:7d:e0:1f:79:39:90:d2:87:4e:d0:dd: af:]
```

The Hash value decrypted by the PC end is:

```
[90:01:50:98:3c:d2:4f:b0:d6:96:3f:7d:28:e1:7f:72:]
```

For the description of how to use TLV structure *stDataPkgIn*, please see sample16.

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below together with the successfully linked *sample_29_hw_md5_lib.LIB* in sample 29 using Keil. Download the successfully linked hex file *sample_30_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef30" and then run PC end program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Senselock EL EXF Program

The code below demonstrates how to generate a random number and encrypt communication returned value with it. The encryption algorithm is quite simple: attach the random number to the end of the returned data and make exclusive-OR operation using the random number and MD5 operation result byte by byte.

```
ret = _rand(1, &output->buff[length]);
...// check ret
for (; i < length; i++)
{
  output->buff[i] ^= output->buff[length];
}
```

Download the successfully linked result *sample_30\hardware\sample_30_hw.hex* to the root directory of Senselock EL device and name it "ef30".

PC End Program

The main program invokes Senselock EL EXF twice. Each time input the same data, but

return with different encryption results. Encrypt the result and print it for comparison. Listed below is the process of decryption: make exclusive-OR operation using the random number attached to the end of returned data against the previous data to restore the real MD5 Hash value.

```
for (i=0; i<stDataPkgOut.len-1; i++)
{
    stDataPkgOut.buff[i] ^= stDataPkgOut.buff[stDataPkgOut.len-1];
}
```

For the way of encapsulating the input data and the basic function framework, please see sample16.

Questions

Q1. How to modify the sample to design more complex communication encryption scheme?

Sample 31: How to use encrypted data files?

Sample Gist

1. When shall an encrypted data file be used?

In some applications, the data file might be more important than the executable file. As we know, If the option "non-readable/non-writable" is chosen in the creation of Senselock EL EXF, there is no way to read it, not even with "developer" authorization. However, if the attacker is able to write his own program into Senselock EL (for example, a disgruntled resigned employee walks away with the developer PIN), the data files might be read by the maliciously EXF. In view of security, sometimes it might be necessary to encrypt and store data files, which shall also be decrypted before they are used in the dongle.

Attention: This sample contains "developer" level operation codes. At the time of software release, make sure not to include them in the client end software!!

Sample Introduction

This sample demonstrates how to use encrypted data files in Senselock EL. Step 1: encrypt the plain text data at the PC end using the software DES_ECB and then write the encrypted files into the dongle under developer authorization. Step 2: run Senselock EL EXF program, decrypt the encrypted file in the dongle and make data processing with the decryption result. Attention: PC end and EXF program share the same DES mode and secret key value.

The EXF functions written into Senselock EL are as follows:

Input: None

Output: Submit the content of encrypted data file whose ID is SEC_FILE_ID (0x0132) to DES_ECB decryption, accumulate all the bytes of the decryption result and finally return the cumulative sum.

For example, if the original plain text file at the PC end contains data like 11 22 33 44…88 (8 data).

Invoke the EXF program to decrypt the encrypted data file and output the data summation value: 396

Sample Implementation

Compile and link the Senselock EL EXF program as mentioned below with Keil. Download the successfully linked hex file *sample_31_hw.hex* using Senselock EL Development Test Tool to the root directory of Senselock EL and name it "ef31" and then run PC End Program. As for how to use the Keil IDE, please see Chapter 5 of *Senselock EL Developer Guide*.

Public Header File

The public header file specifies the encrypted data file ID:

```
#define SEC_FILE_ID 0x0132 // cipher data file ID
```

It also defines the following sub-function number. This sample uses only the sub-function number:`CMD_USE_SEC_FILE` while retaining others for function extension in the future.

```
/* command group definition */
#define CMD_CREATE_SEC_FILE 0x00  // for future use
#define CMD_USE_SEC_FILE 0x01
#define CMD_UPDATE_SEC_FILE 0x02 // for future use
```

Senselock EL EXF Program

The program uses the encryption/decryption algorithm in DES/TDES ECB mode as mentioned in sample 25. Despite the fact that this sample uses only one of the algorithms: DES_ECB, we have included a complete `enc_dec_ecb()` program for better scalability.

The code below fulfills the function of reading the encrypted data file, making DES_ECB decryption, getting data length and working out the sum of all data.

```
case CMD_USE_SEC_FILE:
    ret = _open(SEC_FILE_ID, &hFile);
    ...// check ret
    ret = _read(hFile, 0, 1, &length);
    ...// check ret
    ret = _read(hFile, 1, length, tmp_buf1);
    ...// check ret
    ret = enc_dec_ecb(DES_DEC_ECB, key, tmp_buf1, length, tmp_buf1,
                          &tmp_length);
    ...// check ret, if succeed, tmp_buf1 contains the plaintext
    for ( i=0; i<tmp_length; i++)
    // use tmp_buf1 here, for example: calculate their sum
    {
        tmp_sum += tmp_buf1[i];
    }
    length  = sizeof(tmp_sum);
    CC_TO_LE32(&tmp_sum);
    ret = _mem_copy(output->buff, &tmp_sum, length);
    ...// check ret
    _close(hFile);
Break;
```

Download the successfully linked result *sample_31\hardware\sample_31_hw.hex* to the root directory of Senselock EL device and name it “ef31”.

PC End Program

While running, the main program will offer the operator a menu:

```
1.Write DES-encrypted data file into Senselock EL[dev mode];
2.Use data file in Senselock EL to do some calculation [user mode];
3.Exit;
Your choice:
```

Choose '1' to make DES_ECB encryption of the *sample31\win31\plaintext.txt* at the PC end using software algorithm and write it into Senselock EL under developer authorization using `S4WriteFile()`. For the description of how to use the software algorithm library *Cryptst.lib* at the PC end, please see Appendix A: Description of Cryptographic Algorithm Usage of "*Senselock EL Developer Guide*".

Choose '2' to invoke Senselock EL EXF and make DES_ECB decryption of the encrypted data file written in the last step and output the sum of decrypted results.

Choose '3' to exit the program.

Attention: The PC end and Senselock EL EXF must use the same DES secret key, in this sample, the key below is employed:

```
unsigned char key[8] = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07};
```

Questions

Q1. If the data files are very important, yet you do not like to take the trouble to encrypt them in the dongle, what precautions shall be taken to protect their security?

7. Tips (Answers) to Questions

Given below are the tips (answers) to the questions in this document. Some answers are close-ended while others are open-ended. What we are very pleased to see is that readers can find better solutions to these questions than the ones presented here.

Sample 2

Q1. What are the difference and relationship between self-defined dongle ID, customer ID and unique serial number of the hardware?

"Self-defined Dongle ID" is 8-byte data defined for Senselock EL by the software developer himself. It can only be created and modified under development level authorization;

By default, "Customer ID" is 4-byte all-zero data. Senselock can provide customers with setting "Customer ID" service. Once it is set, Senselock can guarantee the uniqueness of "Customer ID". This value is unchangeable after delivery;

"Unique serial number of the hardware" is the global unique serial number in relationship with the smart card chip. The uniqueness is guaranteed by the chip manufacturer. If the software developer has not set "self-defined dongle ID", after invoking `S4Enum()` to list specific Senselock EL devices, the 8-byte ID that the member variable *bID* of the device's context structure contains is the "unique serial number of the hardware"; otherwise the value is "self-defined dongle ID" set by the developer.

Q2. If the self-defined dongle ID is set, is the unique serial number of the hardware still useful? How to get the unique serial number of the hardware?

It can still be used. This value can be obtained via `S4Control (…S4_GET_SERIAL_NUMBER…)`.

Sample 4

Q1. If devices in HID mode and in USB mode are plugged in the PC concurrently, can they operate normally at the same time?

Senselock EL devices in HID mode and USB mode can operate concurrently.

Sample 6

Q1. In the example as mentioned before, to create a subdirectory "df21" under "df11", can we set the size of new directory as the "remnant space" (9866 bytes) of "df11"?

No, because creating a new directory will also seize 42 byte space of the parent directory for the sake of management.

Sample 8

Q1. What role does the user PIN play? Why not emphasize its security?

User PIN is usually included in the software sent to end users. In principle, there is no "security" issue in the user PIN, with which the software developer can simply check whether it is a dongle of his own.

Q2. Why is it not recommended to differentiate your own devices using user PINs in the event of multiple Senselock EL dongles?

Since there is no “security” issue, it is very likely that many Senselock EL dongles have the same user PIN (default value).

Sample 11

Q1. Suppose independent software protection module is stored in different subdirectories. Will the acquisition of the developer authorization for parent directory endanger the security of the contents in the subdirectory?

The acquisition of the developer authorization for the parent directory does not allow the acquisition of the developer authorization for the subdirectory concurrently. Of course, the developer authorization for the parent directory can be used to delete the subdirectory and thus damage the software, but obviously this situation will not impact the security of the contents in the subdirectory.

Sample 14

Q1. How to open the device in an “exclusive” mode?

Open the Senselock EL device with `S4OpenEx()`, using `S4_EXCLUSIVE_MODE` as member variable of the structure pointer parameter `pS4OpenInfo`.

Sample 15

Q1. Under what situation is a problem very likely to occur if Senselock EL is operated in multi-thread/multi-process but without using the synchronization mechanism?

When multiple threads/processes operate in different directories, directory switching will cause the loss of security status in previous directory, thus impacting the subsequent operations.

Sample 16

Q1. What are the disadvantage of transporting data with the structure?

The relative stability of the data format will not only ease program composition, but also data analysis by the cracker. This is so called “One cannot serve God and Mammon”. Security and convenience are always contradictory.

Q2. If the structure in this sample is not used, how to handle input/output to accomplish the same purpose?

Omitted.

Q3. Under what situation is byte sequence conversion required?

If there are different Big Endian/Small Endian modes for the PC end and hardware programs and when the input/output data are of multiple bytes, byte sequence conversion is required.

Sample 17

Q1. What are advantages and disadvantages of using Senselock EL code memory in comparison with Senselock EL data files?

For the advantages of Senselock EL code memory, see the "Sample Gist" part of the sample. The major disadvantage is when code memory is used to store large data tables, it is necessary to re-compile/re-link and re-download the executable file rather than simply replace independent data file in the event of data replacement or update.

Q2. In terms of some specific applications, what kind of data can be put into Senselock EL code memory?

Omitted.

Sample 18

Q1. Input data more than 250 bytes by several times using the shared memory. Under what situation can this feature be used?

Considering about the case when the data in excess of 250 bytes shall be processed all at once in the dongle (rather than by batch).

Q2. Senselock EL EXF under different directories (or the same directory) can also use the shared memory. Under what situation can this feature be used?

Communication and coordination of multiple Senselock EL programs.

Q3. How to achieve the goal of making several calls to Senselock EL which are mutually related using the shared memory?

A complete example will be given in the future sequel.

Sample 19

Q1. What problems might arise if different memory modes are selected for the hardware function library and the Senselock EL executable program?

Program linking will fail.

Sample 20

Q1. What is the difference between invoking the function for updating data file update_internal_file() and invoking directly S4WriteFile（S4_UPDATE_FILE）in this sample?

Difference in execution authorization: one is user level, the other is developer level;

Any internal executable file which is created using `S4_FILE_EXECUTE_ONLY` flag can not get updated through the former, it can only be updated through the latter;

Contrarily, any file created by SES function `_create()` can only get updated through the former.

Q2. If no other restrictions are imposed, what are the huge hidden security troubles in this sample?

If `update_internal_file()` or `read_internal_file()` is used to write/read the executable file/data file (if the executable file is not flagged with `S4_FILE_EXECUTE_ONLY` when it is created), there will be critical hidden security trouble!!! As for the private key file, since the private key file is not readable under any circumstances, there will not be trouble of private key read-out leakage, but they might be overwritten.

Q4. What measures shall be taken to eliminate these hidden security troubles?

In actual use, the input data to be written into the dongle shall be encrypted (just as what they are used in our remote update platform). Furthermore, unless encrypted, the read files shall be strictly limited inside the dongle.

Sample 21

Q1. How to judge the type of a file in the dongle using `_create()` in the running of a Senselock EL EXF?

Invoke `_create (destination file ID, any size, tested file type (e.g. SES_FILE_TYPE_EXE), CREATE_OPEN_EXISTING)`. If `SES_SUCCESS` is returned, it indicates the tested file type is correct; if the file exists and the tested file type is incorrect, `SES_FILE_ACCESS` will be returned. `SES_FILE_ACCESS` error[2] will not be returned in other situations.

Q2. How to achieve the same goal using Senselock EL API?

Invoke `S4Control(…S4_EF_INFO…)`, and get the type of the test objective file from the returned structure.

Sample 22

Q1. If double d1 = 2 and float f1 = 2 are defined in VC, what are their IEEE representation? How are they stored in the PC end memory? After they are transported into Senselock EL, can they assign value directly to the `DOUBLE_T` structure variable or float variable?

[2] Another special situation: if the run EXF is VM executable file and the destination file exists and is an XA executable file, in this case, even if `SES_FILE_TYPE_EXE` is used as file type parameter, `SES_FILE_ACCESS` error will also be returned. Considering it is a rare situation, we just want to give a small reminder here and ignore it.

The storage representation can be viewed in the memory window under the VC debug mode. The double type at the PC end can assign value directly to `DOUBLE_T` structure of Senselock EL; the float type cannot assign value to float variable of Senselock EL till after undergoing byte sequence conversion.

Sample 25

Q1. In the sample, what are the maximum bytes of data to be encrypted that the encapsulated functions: `enc_dec_ecb()` and `enc_dec_cbc()` at the PC end can process at once?

Both `enc_dec_ecb()` and `enc_dec_cbc()` attempt to complete encryption/decryption in the one-time communication with Senselock EL. Limited by the size of the communication buffer, the data to be encrypted in `enc_dec_ecb()` shall not exceed 248 bytes. As `enc_dec_cbc()` needs additional 8-byte initial vector and 1-byte length data (defined both in DATA_BLOCK), it can process fewer plaintext data. The length of plaintext shall not exceed 248-9 = 239 bytes.

Q2. How to modify them so that they can process more data to be encrypted?

Transport the data by batch into the dongle for processing. The detailed implementation process is omitted.

Q3. When the length of the input data is not the integer multiple of 8, how does the function `enc_dec_cbc ()` in EXF program of this sample do padding? Are there any other substitute padding schemes? What are the advantages and disadvantages of each padding scheme?

`enc_dec_cbc()` groups the input data by 8 bytes. The remnant less-than-8-byte part fills in full zero to 8 bytes. If necessary, you can also fill in all 0x80, or all 0xFF, or length+padding value. The advantages and disadvantages of each padding schemes depend on whether restoring the original data according to the decrypted padded data iss convenient and practical.

Sample 26

Q1. If the SES functions `_rsa_sign()` and `_rsa_veri()` must be used directly for RSA data signing/signature verification, how to achieve this? Are there any inconveniences?

For the shortcomings of the above two SES functions, please see the “Sample Gist” part in this sample. Detailed implementation is omitted.

Q2. How to modify the functions `make_signature()` and `verify_signature()` in EXF so that they can support more Hash algorithms?

Omitted.

Sample 27

Q1. How to achieve HMAC-MD5 with reference to this sample and sample 29?

Omitted.

Sample 28

Q1. How to make the functions in sample 25 into a hardware function library so that this sample can use its CBC encryption functions directly?

Omitted.

Sample 30

Q1. How to modify the sample to design more complex communication encryption scheme?

Omitted.

Sample 31

Q1. If the data files are very important, yet you do not like to take the trouble to encrypt them in the dongle, what precautions shall be taken to protect their security?

Make sure to protect the developer PIN!!