# Elite EL Developer Guide

**Nov 2010**

**Version 3.0.0.7**

SENSELOCK

Reliable Professional Partner

# License Agreement

This document is a legal agreement between you (either an individual or an entity) and Senselock Software Technology Co.,Ltd. ("Senselock"). If you are not willing to be bound by the terms of this agreement, you should promptly (and at least within 3 days from the date you received this package) return the unused developer's kit and the programmer's guide to Senselock. Use of the software indicates your acceptance of these terms.

GRANT OF LICENSE. The Software is being licensed to you, which means you have the right to use the Software only in accordance with this License Agreement. You may (a) copy the software for your internal use, (b) modify the software for the purpose of integrating the software with your application and (c) merge the software with other programs. You may also (a) distribute the merged/modified software to your customers provided that you enter into an agreement with them that is substantially in the form of this Agreement to assure that they treat the software as confidential and proprietary and (b) limit the use of the software by your customers with products you sell to them.

NONPERMITTED USES. Without the express permission of Senselock, you may not (a) use, copy, modify, alter, or transfer, electronically or otherwise, the Software or documentation except as expressly permitted in this License Agreement, or (b) translate, reverse program, disassemble, decompile, or otherwise reverse engineer the Software.

Senselock warrants for a period of twelve (12) months after date of purchase its software and the Senselock ELkey as set forth in this Agreement and License. All the responsibilities of Senselock Software Technology Co.,Ltd and all the compensation you can get during warranty period are: you can select to replace/repair your device(s) or accept other remedial measures.

LIMITATION OF LIABILITY. Senselock will NOT in any event be liable for any damages whatsoever arising out of or related to the use of or inability to use the product, including but not limited to direct, indirect, special, incidental, or consequential damages, and damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss.

All the product, including SenseLock EL™ , the software, the document, other material shipped with this product, and backups made by you are copyrighted by Beijing Senselock Software Technology Co.,Ltd.

TERMINATION. Your failure to comply with the terms of this Agreement shall terminate your license and this Agreement.

"SenseLock EL" is registered trademarks of Beijing Senselock Software Technology Co.,Ltd.

All products referenced throughout this document are registered trademarks of their respective proprietors.

# Contact Information

Senselock Software Technology Co.,Ltd®

*Headquarter Office Address:Rm.1201, Blk. B, ZhuCheng Mansion, ZhongGuanCun South Street, Haidian District, Beijing P.R.China*

*Post:100086*

*Tel: +86-10-51581366*

*Fax:+86-10-51581365*

*Sales representative:* sales@senselock.com

*Tech service:* tech@senselock.com

*Website*:http://www.senselock.com/

# Index

# 1. Product Introduction

Senselock EL is a new generation software dongle developed by Senselock Software Technology Co.Ltd. It is now the most secure software protection product in the world.

Two key technologies guarantee software protection truly:

- Program code protection

  You can transfer approximately 10,000 lines of High Level Language Program codes to Senselock ELand execute them there. These transferred codes will not leave any trace on the computer, nor can anyone get a copy of these codes. Your software can run on two systems: one is the computer; the other is SenseLock EL. Piracy of the software protected by Senselock ELis impossible because Senselock ELhardware is not copyable.

- Reliable hardware base

  Uncopyability and attack protection are achieved via hardware security. To this end, Senselock ELnot only employs the Smart Card technology, but also selects the best Smart Card chip (which has passed security authentication by CC EAL5+[1]). If you have studied relevant security standards carefully, you will find the hardware of Senselock ELis absolutely trustworthy.

Senselock ELcomes up with the “uncrackable” security model for the first time in the software protection field, so it can put a complete end to piracy, thus protecting maximally the economic benefits of software developers.

## 1.1. New Software Protection Method

If you have used dongles of other types, please try your best to forget the technical details of those dongles and digest the contents of this section, because this is a cutting-edge and brand new notion. If you have never used any dongle before, take a look at the picture below and you will find the whole process quite easy to understand.

Do remember that there are no such concepts as “data area”, “algorithm unit”, etc., which are all outdated. Now your dongle is provided with a complete “operating system” consisting of all files: executable files, data files and key files as well as directories, of course. You can imagine it as a MS-DOS operating system.

Next let us take a look at how software protection is achieved (Figure 1-1).

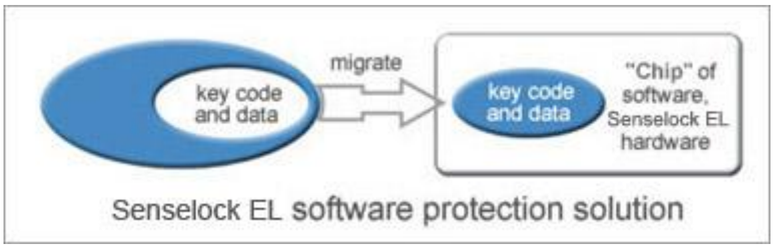

key code and data
migrate
key code and data
"Chip" of software, Senselock EL hardware
Senselock EL software protection solution

Figure 1-1 Senselock ELSoftware Protection Method

1. Develop and debug software wholeheartedly, ignoring completely the dongle and software protection technologies;

[1] This is a very high security certification level. For details, please refer to the contents of ISO15408 standard.

2. Find some relatively important codes from the software and compile these codes via the compiler compatible with Senselock ELbefore writing into SenseLock EL;

3. Add the call of Senselock ELto where the codes are “dug out” in the software;

4. When the software is run, these ported codes will run inside Senselock ELand return the results to the software;

As part of the software is ported to the hardware of SenseLock EL, the software will be incomplete if it is detached from SenseLock EL. As a result, it is a “mission impossible” to crack the software protected by SenseLock EL.

Each ported code part can become an independent “executable file” inside SenseLock EL. Of course, they can also be combined into one. Senselock ELcan now offer a maximum of 64K security storage space with almost no limit over the number of files (the file header may take up some space, so too many files will result in a waste of storage room). The total number of portable codes is 10,000 lines or so, so Senselock ELcan protect as much software as it can be.

Attention: Despite the fact that code “porting” is the most fundamental software protection method of SenseLock EL, this does not mean you need to discard totally the original technical strategy. Under most circumstances, they are reciprocally complementary, yet it is imperative to remember: your software is secure in a real sense unless the codes are ported into SenseLock EL.

## 1.2. Basic Product Composition

Senselock ELis not a simple dongle, so you need to develop “dongle” version for some parts of the software on its operating system[2]. This is not an arduous task. Under most circumstances, you can make them run in Senselock ELby very simple modification of the codes since Senselock ELsupports codes compiled with standard C language. If the software is developed via other high-level languages, such as Delphi, you need to re-write the ported part in C language. Although some efforts are required for the “translation” between high-level languages, if “translating” several hundred-line codes can keep you thoroughly away from piracy, it is definitely worth doing. In the subsequent part of this text, we usually call the codes ported into Senselock EL“Senselock ELApplication” or “Senselock ELEXFs”.

The Senselock ELproduct is comprised of the following parts:

- Senselock ELdevelopment tool kit
- Senselock ELhardware
- Senselock ELuser running environment

### 1.2.1. Senselock ELDevelopment Tool Kit

The most important part is the IDE (integrated development environment) of Senselock ELEXF. Senselock ELsupports standard C language development. Any compiler that supports MCS 51 or SmartXA 2 can be used to develop Senselock ELEXFs. Our SDK contains a full-function compilation environment Skit. If you are more used to the trendy development tools, we recommend C51 compiler by Keil Software or RKit by Raisonance SA to you. You can also visit *http://www.keil.com* and *http://www.rainsonance.com* to get information about these two kinds of compilation software.

Some tool software is also included in the development tool kit. They are

[2] The internal operating system of Senselock ELcomplies with the international standard of the Smart Card operating system. You can get relevant information about “Smart Card Operating System” by consulting ISO7816 serial standards.

- Development test tool. Assisting the development and testing of codes ported to Senselock EL and conducting simple management of the equipment;
- User test tool. It can be distributed to software ultimate users, helping them to address the possible configuration fault;
- Batch setup tool. A high-efficiency dongle batch initialization tool;

You can find plenty of sample codes in the development tool kit as references or templates in the middle of development.

Attention: For the details of the compiler, please refer to Chapter 5 " Direction for Compiler Usage" ; for the details of the tool software, please refer to Chapter 6 " Instruction for Tool Software Usage" .

### 1.2.2. Senselock EL Hardware Device

Developed fully based on Smart Card technology and employing 16-bit, 16MHz high-performance CPU, the hardware device is the running environment and storage media for Senselock ELEXFs. Senselock EL is offered mainly as a USB interface device. As an option, it can also be offered in the form of standard Smart Card. Senselock EL is a highly integrated product whose devices (including CPU, RAM, EEPROM and USB communication module) are integrated in the same Smart Card chip, thus enhancing tremendously the security and stability of the product.

There is a global unique serial number for each Senselock EL device. This serial number is already specified at the stage of chip production.

### 1.2.3. User [3] Running Environment

This is chiefly the driver of Senselock EL device. Senselock EL can run in two modes: one is to access the device via standard USB driver. This is what we call standard mode. The other is no-driver mode. In this case, it will be recognized by the Windows operating system as a standard HID device. Compared with the standard mode, the no-driver mode needs to sacrifice some advanced functions and versatility (such as multi-process access to the device), though it is easy to operate, so we recommend the standard mode.

## 1.3. Important Security Precautions

Before getting started with software protection, please make sure you have already understood the contents of this section!

### 1.3.1. PIN

Two levels of password are available in SenseLock EL: developer PIN and user PIN (The password is sometimes also called "PIN code" in the text). Different operation authorizations[4] can be obtained using different levels of PIN.

The developer PIN is 24 bytes. After the PIN is successfully authenticated, a user can get the setup authorization for Senselock ELdevice. Although the possession of the developer PIN does not allow for the direct reading of the data in the Senselock ELhardware, the disclosure of the

[3] The "user" in the text refers to ultimate software user.

[4] Operation authorization is sometimes also known as "security status". If the hardware device is reset or powered off, the operation authorization will be lost. For example, after the computer sleeps, the software can continue to run after it is restarted, but since the USB port has undergone the power-off process, the operation authorization of the dongle will be lost, thus causing failure in operating the dongle.

developer PIN will threaten critically the security of data inside SenseLock EL. When Senselock ELis used for the first time, the default developer PIN is “123456781234567812345678” (`0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38`). Before Senselock ELis released, make sure to modify the default developer PIN to a secret value.

The user PIN is 8 bytes. The default value of the PIN is “12345678” (`0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38`). The successful authentication of the user PIN can only get the “call authorization” for the Senselock ELdevice. In other words, a user can call the EXFs in SenseLock EL, but s/he is denied of access to other files such as data file. In principle, there is no “security” problem with the user PIN. You can use it to check simply whether it is your dongle.

Given below are some important security details in relation to the PIN:

- Both the developer PIN and user PIN are designed for the directories in SenseLock EL, but not for specific Senselock ELdevice. Each directory contains its own developer PIN and user PIN. The security attributes between directories are independent, so modifying the PIN of the root directory does not mean the PIN of the sub-directory is modified accordingly. In the like manner, the acquisition of root directory authorization will not allow for the acquisition of relevant sub-directory authorization automatically. This feature allows several irrelevant software to be protected by the dongle without reciprocal interference and threat, but deleting the root directory will cause the contents of the sub-directory to be deleted in the meantime.
  Unless it is necessary, we do not recommend the use of multi-level directory to avoid chaos in management.
  For detailed description of the directory, please refer to the next section “File System”.

- If there are 15 continuous re-trial errors for the developer PIN of the root directory, the hardware device will be locked. Once it is locked, there is no way to recover it, so it is imperative to keep the developer PIN of the dongle in a safe place and please do remember that Senselock cannot recover your PIN or unlock the device. This is to protect your security to an maximum extent.
  The re-trial error in sub-directory developer PIN will only cause the directory to be locked, but it can still be deleted by wiping the upper-level directory.

- No matter how many trials are made for user PIN, the device will never be locked.

- Whenever the file system is re-initialized, such as deleting and re-creating a root directory, clearing the sub-directory, the two-level PINs of the directory will be both restored to default values, so do remember to modify the developer PIN to a secret value after the file system is re-initialized.

### 1.3.2. File System

Getting to know some security details regarding the file system will help create a more secure protection solution.

Senselock ELsupports the “directory-file” structure. The internal storage space of Senselock ELis managed by directory. Given below are some key features of Senselock ELdirectories:

- There is one and only one root directory, below which there are sub-directories, but the maximum levels of directory are three;

- To create a directory, the space occupied by the directory shall be specified. Once the directory is created, the size of it is unchangeable. The root directory takes up all the storage space of the hardware.

- The root directory can be deleted. When the root directory is deleted, all the contents in the directory, such as sub-directories and files will be deleted as well.

- Any directory's parent, sibling and child directory must not share the same directory ID.

- The sub-directory can only be cleared, but not deleted directly. Clearing the sub-directory will delete all the contents in the directory, such as sub-directories, files, etc. After the sub-directory is cleared, its developer level PIN and user level PIN will be restored to default values. To wipe out the sub-directories to release all seized space, it is necessary to clear/delete their upper-level directory.

- Net dongle only has root directory.

In SenseLock EL, there are usually three kinds of files: executable files, data files and key files, none of which can be read directly from the hardware. Files can be created and modified after the developer PIN is authenticated.

- Executable file

The executable file is a kind of file format that can be loaded and run on the internal operating system of SenseLock EL. The ported codes will be compiled into executable files in the middle of software protection. There are two kinds of executable files: VM executable file (also known as general executable file) and XA executable file, corresponding to two different running modes. For the detailed description of the running mode, please refer to Chapter 4 "Developing Senselock ELCodes". XA executable files are supported only by Senselock EL whose hardware version is 2.3 or above. They surpass VM executable files in running efficiency, but they show poorer versatility in SenseLock EL. By running the tool software available in the product package, you can get the hardware version number of SenseLock EL.

An executable file can operate other files in the same directory through the system function in the hardware, such as reading and writing data files. When it comes to the operation of executable files, the situation is somewhat complicated: it is not necessarily true that an executable file can be read and written by another executable file. It depends on the security attributes of itself (Please see Table 1-1).

- Data file

The data file is designed only to store binary data. The programs on the computer cannot be read directly from the data files in SenseLock EL. To access the data files in SenseLock EL, you need to do it via the EXF in SenseLock EL.

- Key file

Only the RSA key file is supported now. You can consult Chapter 8 "System Function Reference" for more detailed information.

We summarize the security attributes of the internal files commonly used in Senselock EL as follows:

Table 1-1 Security Attributes of Senselock EL Files

| | Developer level (authenticate developer PIN) | User level (authenticate user PIN) | Internal (internal executable files) |
|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Executable file | Unreadable & unwritable | W | E | N |
| | Readable & writable internally | W | E | R/W |
| Data file | | W | N | R/W |
| Key file | | W | N | W |

R: Read W: Write E: Execute N: None

There are more detailed specifications governing the security attributes involved in internal file operation in SenseLock EL. For details, please refer to the concerned file operation part in Chapter 8 “System Function Reference”.

The files in Senselock EL are identified via a 16-bit (2-byte) unsigned integer. Part of the IDs are reserved by the system. They are `0x0000, 0x0015, 0x0016, 0x0018, 0x001e, 0x3f00, 0x3f01, 0x3f02, 0x3f03, 0x3f04.`

## 1.4. More Functions

Senselock EL contains the hardware co-processor of RSA (1024 bits) and DES/TDES cryptography algorithm and offers SHA-1 algorithm function and hardware true random number generator. Based on these cryptography functions, Senselock presents a remote updating and sales management platform. The software protected by Senselock EL can be billed and updated via the Internet. By this means, the management of software sales channels can be carried out. As Senselock opens the bottom-layer technical details for remote updating, you can create your own remote updating and sales management platform using the bottom-layer technology offered by Senselock.

Using the cryptography algorithms of SenseLock EL, you can also log on the Windows operating system, encrypt and decrypt email and perform various kinds of complex network security functions.

# 2. Product Installation

This chapter offers a guide on how to install SDK of SenseLock EL.

## 2.1. System Requirements

To use SDK of SenseLock EL, it is recommended to use the PC that runs the operating system of Windows 98, Windows2000 or above. If it is installed on NT4, you need to install Service Pack 4, IE4.0 or above first.

Senselock EL supports Microsoft Windows, Linux and Mac OS X. You can use Senselock EL device on the above operating system.

## 2.2. SDK Installation

Put the CD into the CD-ROM. If the installed program is not started automatically, please run *Windows\Setup.exe* in SDK CD of Senselock ELand complete the installation according to the prompt of installation wizard.

After it is successfully installed, the target directory will contain the following:

| Sub-directory | Contents |
|---|---|
| *API* | API function of SenseLock EL. |
| *Doc* | Product development manual and its description document. |
| *Driver* | Hardware device driver. For the details about the installation of the driver, please refer to 2.2.1 that “About Driver Installation” and Appendix E ”Reference to Driver Installation”. |
| *IDE* | The support library required when Senselock offers integrated development environment (such as Keil C51).<br>Senselock EL hardware system function library. For details, please refer to Chapter 8 “System Function Reference”. |
| *Samples* | Sample codes, including basic operational demos and cases’ source code |
| *Tools* | Various kinds of tool software. For the details about the usage of tool software, please refer to Chapter 6 “Direction for Tool Software Usage”. |
| *Support* | All kinds of supporting resources, such as cryptography algorithm library and diagnose tool. For details, please refer to Appendix A “Direction for cryptography algorithm usage”. |

### 2.2.1. Direction for Device Driver Installation

The device driver of Senselock EL will be installed automatically in the installation of SDK. Under some circumstances, you might need to install the driver independently (if you use SDK directly by copying it to other computer or release your software). In this case, you need to use the driver installation tool available in SDK.

Located under the directory of *%SDK%\Driver*, this tool is called “*InstWiz3.exe*”. This is a WIN32 EXF, which can accept the following command line parameters:

- `/INSTALL` install driver (this is a default option)
- `/UNINSTALL` uninstall the driver
- `/NOGUI` Installation interface not displayed

These command line parameters can be used in combination. For example, you can run “`InstWiz3.exe /uninstall /nogui`” to uninstall the device driver in a hidden way.

This tool can also be used to check the installation status of the device driver of Senselock ELin the current system: run *InstWiz3.exe* and the tool will check and report the installation status of the driver automatically.

Note:

The InstWiz3.exe cannot be executed alone. Under the same directory, these files (mkSetup.dll, language.dll) and directories (win98, winlh64, winlh86, winxp64, winxp86) must be consistently unchanged remaining original file structure.

There are several ways to check whether the driver is successfully installed:

1. Insert Senselock EL device to check whether Senselock Senselock ELv2.x shows up normally in the hardware device list ("Device Manager"→ "Smart Card Reader"). For Windows98, Window2000 or above, if the device is inserted for the first time, the system will give a prompt of "new device found" automatically and complete the installation;

2. Insert Senselock EL device and run the development test tool or user tool to check out whether access to the device is successful;

If you are quite familiar with the installation process of the device driver, you can also choose to install the Senselock EL device driver manually. The directory *%SDK%\Driver\Obj* contains all the files and directions required for the manual installation of the driver.

If you want to complete the installation of the device driver on your own via API, please refer to Appendix E "API Reference for Device Driver Installation".

The device driver of Senselock EL has passed Microsoft's logo authentication (Designed for Windows XP). For details, please visit http://testedproducts.windowsmarketplace.com/ and query the product SenseIV.

## 2.3. Uninstall SDK

SDK can be uninstalled directly from "Control Panel"→"Add / Delete Program". The device driver of Senselock ELwill be deleted automatically while SDK is being deleted.

# 3. Preparation for Use

This chapter will use a sample to exemplify how to port part of the codes into SenseLock EL. It will help you understand roughly the total process of Senselock EL software protection.

The full sample is available under the directory of *%SDK%\samples\Developer Manual\Chap3*.

## 3.1. Demo Software

To make the process easier to understand, we have chosen a simple demo program: bubble-sorting algorithm. Before it is ported, the code of the software is as follows:

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.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;
      }
    }
  }
}

/*main procedure*/
void main()
{
  unsigned char test[] = {4,3,8,2,9,7,1,5,0,6};
  int len = sizeof(test);
  int i;

  bubble_sort(test, len);

  printf("result:\n");
  for (i=0;i<len;i++)
  {
    printf("%d ",test[i]);
  }
}
```

Create a new Win32 Console Application in Microsoft Visual C＋＋6 and add the above code to the VC6 project. In this sample, suppose we save the project as *C:\s4demo\Win32\demo1.dsp*, and the above code as *C:\s4demo\Win32\pc_demo1.c*. Compile and run the above code and it will output the following result:

```
result:
9 8 7 6 5 4 3 2 1 0
```

Let’s suppose the sorting algorithm in the software is a code that shall be protected and that shall be ported into “SenseLock EL”.

## 3.2. Code Porting

Before the code is ported, you need to choose a compiler supported by SenseLock EL. In this sample, Keil C51 is selected. This chapter describes only necessary parts for the configuration of the Keil C51 compiler. For more details about the usage of the compiler, please refer to Chapter 5 “Direction for Compiler Usage”.

### 3.2.1. Create a Senselock EL Project

Run Keil µ Vision, select “Project”→ “New Project…” on the menu, name the project as “Demo1.uv 2”, and save it on the hard disk. In this sample, suppose the project is saved as “*C:\s4demo\Demo1.uv2*”.

After that, µ Vision will prompt you to choose device. Select “Intel”→ “8052AH” from the list of “Generic CPU Data Base”. Keil will ask “Copy Standard 8051 Startup Code to Project Folder and Add File to Project?”, select “Yes”.At this stage, a blank project is successfully completed. Some additional configurations shall be made before the project can be supported by “SenseLock EL”.

Select “project” → “Options for Target1” from the menu. On the “Target” page, set “Memory Model” as “Large: variables in XDATA”, and then switch to the “Output” page, choose “Create Hex File”.

### 3.2.2. Compile Senselock EL Codes

Now we can add the ported code to the project. Create a C file called *demo1.c* and copy the bubble-sort () function in *pc_demo1.c* to *demo1.c* as well as adding the main () function of SenseLock EL. The full contents of *demo1.c* are given below:

```
#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 ELmain procedure*/
void main()
{
  unsigned char *test = pbInBuff;
  int len = bInLen;

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

The above compiled code can be run in SenseLock EL. We make a brief description of the code before getting started with compilation.

The function of the code in this section is to submit the data and data length (byte count) received at the PC end to the `bubble_sort ()` function of Senselock ELfor processing and return the processing results to PC.

The *`pbInBuff`* and *`bInLen`* are two pre-defined macros. The former points to communication buffer, namely, the data sent by the PC while the latter indicates the length of transmitted data.

The code of `bubble-sort` () function is completely identical with that in *pc_demo1.c*. This is a very important feature of SenseLock EL: supporting the porting of standard C code.

`_set_response ()` and `_exit ()` are two system functions of the SenseLock EL. The former is designed to transport data to PC while the latter returns from the application of SenseLock EL. For the detailed description of the two system functions, please see Chapter 8 "System Function Reference".

For the sake of convenience, *demo1.c* is saved to the directory of *C:\s4demo*.

## 3.2.3. Compile and Test Senselock EL Codes

Before modifying the code (*pc_demo1.c*) at the PC end, you can compile and test the Senselock EL code you have just compiled.

In the just-created project, choose the menu "Project"→ "Components, Environment and Books" and click "Add Files" in the dialog box to add *demo1.c* to the project. In the meantime, you can add "*Ses51L.lib*" offered by Senselock EL SDK to the project[5].

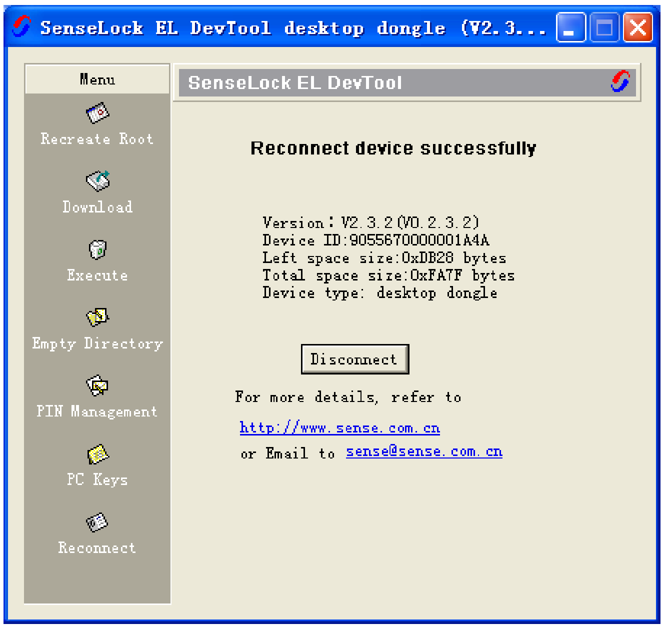
Choose the menu "Project"→"Build target" to compile codes. If the compilation is successful, a file (*demo1.hex*) will be generated under the directory of *C:\s4demo*. This file is the result of code compilation.

Now you can test or debug Senselock EL codes. For the details about testing and debugging technique, please refer to Chapter 4 "Developing Senselock ELCode" and Chapter 5 "Direction for Compiler Usage".

## 3.2.4. Download Codes to Senselock EL Hardware

Connect SenseLock EL2.3 to the computer and run the "development test tool (*%SDK%\Tools\Devtest.exe*) available in SDK. If the device driver is correctly installed, the tool will display the global unique serial number for the connected device. Click "Reconnect" to view the detailed information about the hardware as shown in Figure 3-1.

[5] It is recommended to copy *`ses51L.lib`* and *`ses_v3.h`* to the working directory, namely *`C:\s4demo`* in this sample.



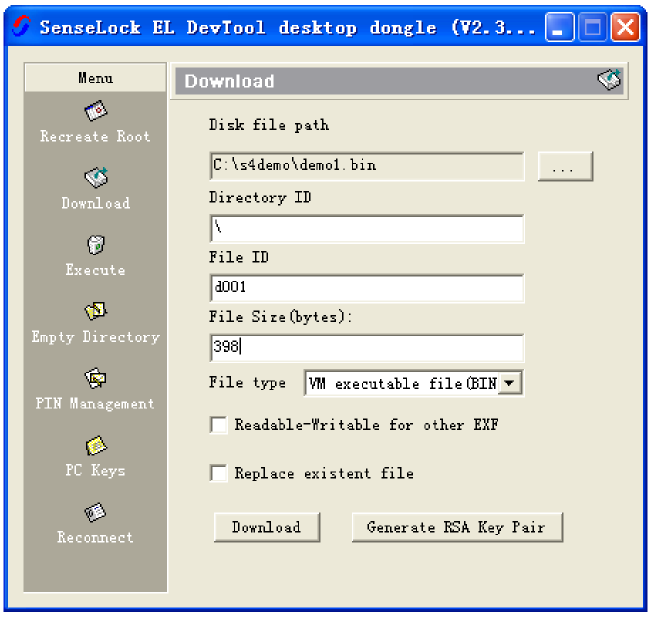
Reconnect device successfully

```
Version: V2.3.2(V0.2.3.2)
Device ID:9055670000001A4A
Left space size:0xDB28 bytes
Total space size:0xFA7F bytes
Device type: desktop dongle
```

For more details, refer to
http://www.sense.com.cn
or Email to sense@sense.com.cn

Figure 3-1 Senselock ELDevelopment Test Tool — Hardware Information Display

For the convenience of testing, we first re-initialize the device and click “Recreate Root” to create the file system in SenseLock EL. Suppose we download the compiled file *demo1.hex* to the root directory. The file ID is `0xd001`. Click “Download” and fill in the file information according to Figure 3-2.



Disk file path: C:\s4demo\demo1.bin
Directory ID: \
File ID: d001
File Size(bytes): 398
File type: VM executable file(BIN
[ ] Readable-Writable for other EXF
[ ] Replace existent file

Figure 3-2 Download File to Hardware

The software will calculate automatically the size of the file to be created. In this sample, the file shall be at least 398 bytes. You can input a bigger value so that when the file grows bigger, the created file can still be used. To write data to the existing file, simply choose the “Overwrite original file” box.

### 3.2.5. Modify PC Codes

Since `bubble_sort()` function is ported to SenseLock EL, we cancel or delete it in *pc_demo1.c*. In the meantime, we change the called part of `bubble_sort()` in `main()` function into the calling of codes in SenseLock EL. The contents of the modified *pc_demo1.c* are as follows:

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include “sense4.h”

/*bubble sort function removed. Add Senselock ELinvoking code. */
void call_sense4(char *, unsigned char *, int);

/*main procedure*/
void main()
{
  unsigned char test[] = {4,3,8,2,9,7,1,5,0,6};
  int len = sizeof(test);
  int i;

  call_sense4(“d001”, test, len);

  printf("result:\n");
  for (i=0;i<len;i++)
  {
    printf("%d ",test[i]);
  }
}

void call_sense4(char *fid, unsigned char *buff, int len)
{
  /* See remarks for details.*/
}
```

The red, overstrike, big size text indicates the modification made to the original contents. There are altogether three parts:

- Add support for access to Senselock EL hardware device, including header file (*sense4.h*) and corresponding API function library (*sense4.lib+sense4.dll* or *sense4st.lib*)

- Remove the copy of ported codes in the software and add the call function `call_sense4()` to SenseLock EL.

- Add the calling of internal codes of Senselock EL to fulfill the function of ported codes.

For the convenience of reading, the implementation process of the function `call_sense4()` is omitted for the above codes. You can find the detailed information about the function in the notes below. You can see that there is only one modification in the `main()` function.

Notes: About `call_sense4()` function

In this sample, the function of `call_sense4()` is to run the EXF whose ID is 0xd001 in SenseLock EL, taking `test` and `len` as input data and save the operation result to `test`. `call_sense4()` is not a universal API function offered by SDK, but an encapsulation of standard API. The purpose of the function is to enhance the readability of the sample codes.

In fact, to execute the EXF in SenseLock EL, the standard access process should be list devices→open specified device→choose directory→authenticate user PIN→execute…→execute→close the device→release resources. We do not recommend you to access devices using this function in formal software. For the details and suggested method for device access, please refer to Chapter 9 "API Reference" and Chapter 4 "Developing Senselock ELCodes".

The source codes of call_sense4() function:

```
void call_sense4(char *fid, unsigned char *buff, int len)
{
  SENSE4_CONTEXT ctx = {0};
  SENSE4_CONTEXT *pctx = NULL;
  unsigned long size = 0;
  unsigned long ret = 0;

  S4Enum(pctx, &size);
  if (size == 0)
  {
    printf("Senselock ELnot found!\n");
    return;
  }

  pctx = (SENSE4_CONTEXT *)malloc(size);
  if (pctx == NULL)
  {
    printf("Not enough memory!\n");
    return;
  }

  ret = S4Enum(pctx, &size);
  if (ret != S4_SUCCESS)
  {
    printf("Enumerate Senselock ELerror!\n");
    free(pctx);
    return;
  }

  memcpy(&ctx, pctx, sizeof(SENSE4_CONTEXT));
  free(pctx);
  pctx = NULL;

  ret = S4Open(&ctx);
  if (ret != S4_SUCCESS)
  {
    printf("Open Senselock ELfailed!\n");
    return;
  }

  ret = S4ChangeDir(&ctx, "\\");
  if (ret != S4_SUCCESS)
  {
    printf("No root directory found!\n");
    S4Close(&ctx);
    return;
  }

  ret = S4VerifyPin(&ctx, "12345678", 8, S4_USER_PIN);
  if (ret != S4_SUCCESS)
  {
    printf("Verify user PIN failed!\n");
    S4Close(&ctx);
    return;
  }
  ret = S4Execute(&ctx, fid, buff, len, buff, len, &size);
  if (ret != S4_SUCCESS)
  {
    printf("Execute Senselock ELexe failed!\n");
    S4Close(&ctx);
    return;
  }

  S4Close(&ctx);

  return;
}
```

Copy *sense4.h* and *sense4st.lib* in SDK to the directory of *c:\s4demo\win32*, add it to the project and re-compile the VC6 project.

### 3.2.6. Test Protection Results

After taking the above steps, we have completed the protection of sample codes. Insert the configured Senselock EL dongle to the computer, run the re-compiled sample software and you will get the correct output result:

```
result:
9 8 7 6 5 4 3 2 1 0
```

Remove the Senselock EL dongle and re-run the sample software and you will get the prompt for error.

```
Senselock EL not found!
result:
4 3 8 2 9 7 1 5 0 6
```

## 3.3. Program Structure Analysis

Before the demo program is protected, the codes contain two major functions: main function `main()` and function `bubble_sort()`. After it is protected, all the codes contain four major functions: main function `main()`, device access function `call_sense4()`, main function `main()` in SenseLock EL, and function `bubble_sort()` in SenseLock EL. Figure 3-1 illustrates their relationship.

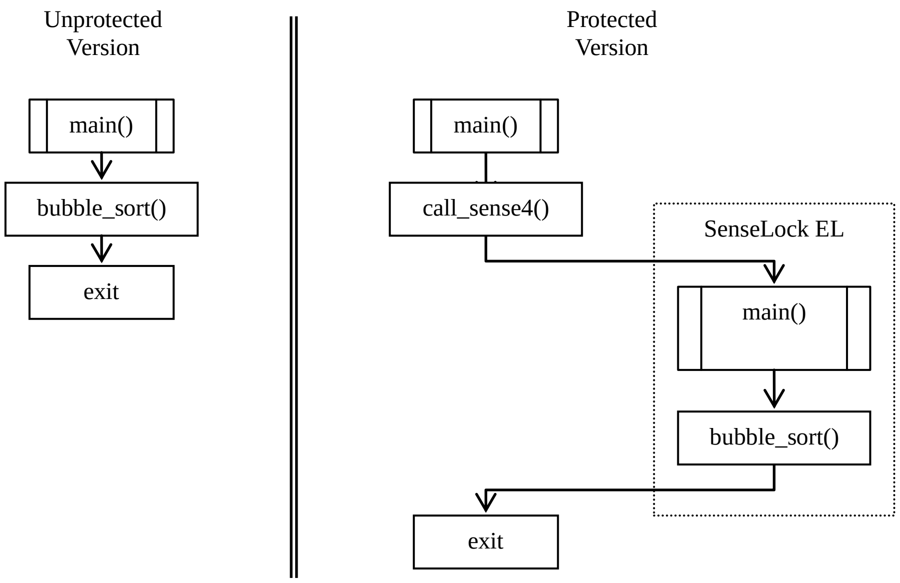


Figure 3-1 Protection Structure of Demo Program

It can be seen that the protected codes seem to be based on the Client/Server structure. Senselock ELis equivalent to a "Service" while `call_sense4()` function and `main()` function in Senselock ELplays the role of communication.

# 4. Develop Senselock EL Codes

You know from the previous chapter that developing Senselock ELapplications involves two tasks: compiling internal codes of the dongle and compiling the software codes at the PC end so as to access SenseLock EL. This chapter will guide you how to compile the internal codes of SenseLock EL, test the codes and operate Senselock ELdevice via API.

## 4.1. Compile Internal Codes of SenseLock EL

Compiling the internal codes of Senselock ELrequires an understanding of the basics of C language. If you have ever developed C programs on the PC, what is covered in this section will be very easy to understand. The C language used for the development of Senselock ELis quite simple, so it does not require an exclusive C programmer.

In the first chapter, we have mentioned there are two different executable files in SenseLock EL: VM executable file and XA executable file, which correspond respectively to two kinds of running modes: virtual machine mode and user mode (also known as VM mode and XA mode in this text). The codes compiled by Keil C51 and Skit provided by us fall into the category of VM files; the codes compiled by Raisonance Rkit fall into category of XA files. For the convenience of description, we call both XA file and VM file internal executable files, abbreviated as EXF (internal Executable File).

If different compiler/running modes are used, there is a slight difference in the requirements for source codes. For details, please refer to Chapter 5 “Direction for Compiler Usage”. We will try best to isolate the sample codes from running modes. Special description will be given for the part in relation to the running modes.

### 4.1.1. Code Structure

The typical structure of Senselock ELinternal codes is as follows:

```
#include “ses_v3.h”

void foo()
{
  /* function code here. */
  return;
}

void main()
{
  /* initialization code here. */
  foo();
  _exit();
}
```

A code that is a standard C language program structure consists of a main function `main()` and several sub-functions. The main function is the entry point of a program where the codes start to be executed. The description of others is given below:

- `#include “ses_v3.h”`

  *ses_v3.h* contains the definition of the extension service of Senselock ELsystem. You must include this file in all codes. SES (System Extension Service, also called Senselock ELsystem function in the text) offers common system functions, such as file operation, clock operation, input & output, etc. All the system functions begin with the underline “_”. For detailed description of SES, please refer to Chapter 8 “System Function

Reference".

- ♦ Program Termination

  You can add system function `_exit()` to any part of the program to terminate the execution of the program. Notice that the use of `return` sentence in main function cannot guarantee the safe exit of the program. Instead, the `_exit()` system function shall be used. This differs slightly from standard C.

### 4.1.2. Input & Output

The purpose of writing internal codes for Senselock ELis to deliver functional service to the software running on the computer so as to protect the software. The communication between them is achieved via "communication buffer", which is a special memory area in the hardware of SenseLock EL. The data, sent from the computer to Senselock ELor from Senselock ELto the computer, will all be saved in the communication buffer.

Two macros are defined in *ses_v3.h* of Senselock ELto receive the data from the communication buffer. One is *`pbInBuff`*, which is a byte type pointer pointing to the communication buffer; the other is *`bInLen`*, which is a single byte integer, indicating the length of data sent over from the computer. Due to the limit of internal resources of SenseLock EL, the communication buffer shall not exceed 250 bytes in size. In other words, the data sent from the computer to Senselock ELor from Senselock ELto the computer each time shall not exceed 250 bytes[6]. To transmit data larger than 250 bytes requires that the data should be partitioned and transmitted several times. For the implementation method, please refer to "Memory Sharing" in this chapter.

When it is necessary to transport data from Senselock ELto the computer, the system function `_set_response()` shall be used. It will copy the data automatically to the communication buffer where they can be received by the computer. It shall be noted that only the last call to `_set_response()` is effective and the previous setting will be overwritten.

The sample as given below illustrates how data are transported between the computer and SenseLock EL.

```
#include "ses_v3.h"
#include "string.h"

unsigned char output[256];
unsigned char input[256];

void main()
{
  int input_len = bInLen;
  int output_len = 0;

  memcpy(input, pbInBuff, input_len);

  /* Operate input data here… */

  /* If we get some output data… */
  _set_response((unsigned char)output_len, output);
  _exit();
}
```

In this sample, the C function `memcpy()`, which is defined in *string.h* is used. The communication buffer itself is also a part of the memory, so you can either copy the contents of it to somewhere else or operate it directly.

[6]In the current version, when data are transported from Senselock ELto the computer, the value can be as big as 252, but there is no guarantee of this value for the subsequent versions.

### 4.1.3. Memory Structure

Senselock ELoffers more than 2K memory space[7], but is still much less than that of PC. In the middle of development, you need to use it cautiously in case overflow might occur.

The memory structure is a complex issue, which has close bearing to the selected compiler. Due to the limited space of this text, it is impossible to talk about all the issues. If you feel it somewhat hard to understand the memory structure of SenseLock EL, you can skip the following details and refer directly to the end of this section "Code Writing Suggestions".

#### 1. Memory Allocation

In the VM mode, the memory is divided into two parts: internal and external RAM. The internal RAM totals 256 bytes while the external RAM is up to 2K bytes. Some compilers might use the internal RAM as stack in their compilation results, so under normal circumstances, try as much as possible to use the external RAM in case the memory may overflow. In the codes, the internal RAM and external RAM are identified respectively by keyword idata and xdata. For example:

```
xdata unsigned long x;
idata unsigned long i;
```

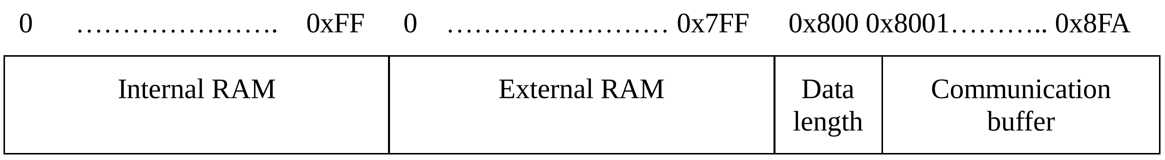
The variable x is defined in the external RAM while the variable i is defined in the internal RAM.

You do not need always to define variables explicitly using these two keywords. The compiler can use the default value. For instance, if the compilation mode is set to Large (which is recommended by us), all the variables without being given a special declaration will be defined in the external RAM. In this case, to define variables in the internal RAM requires that the keyword idata should be used explicitly.

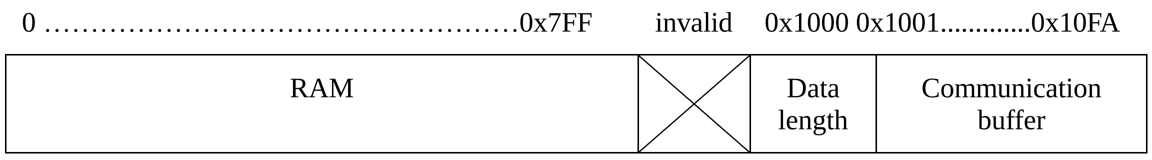
The XA mode does not differentiate internal RAM from external RAM.

The communication buffer is a special memory. Under VM mode and XA mode, there is difference in the memory address of the communication buffer.

The memory structure is shown in Figure 4-1:



a. VM Mode
Memory Structure



b. XA Mode Memory Structure

[7] Under different running circumstances (different compilers are chosen), there are slight differences in available memory. For specific values, please refer to Appendix B.

Figure 4-1 Senselock ELMemory Structure

As for the issue of compilation mode setting, please refer to Chapter 5 “Direction for Compilation Usage”.

## 2. Memory Sharing

For VM mode, unless Senselock ELis reset (for example, the device is unplugged or the S4Control() function is used to send COTROL_RESET_DEVICE control code); otherwise after Senselock ELEXF is executed, the system will not clear automatically the contents of external RAM while the contents of internal RAM will be cleared each time. For XA mode, all the memory will not be cleared automatically.

Using this feature, we can use the memory “that is not cleared automatically” as “shared memory”. This can help share data between different EXFs and can also use them to buffer data among consecutive calls.

For instance, if a program in Senselock ELneeds to receive data larger than 250 bytes, it can be done through a process similar to the following sample:

```
#include “ses_v3.h”

typedef struct {
  unsigned short offset;
  unsigned char len;
  unsigned char buff[1];
} IO_PACKAGE;

DEFINE_AT(unsigned char, big_buff[512], 0x400, RAM_EXT);
IO_PACKAGE *input = NULL;

void main()
{
  input = (IO_PACKAGE *)pbInBuff;
  LE16_TO_CC(&input->len);

  if (input->len != 0)
  {
    memcpy(big_buff + input->offset, input->buff, input->len);
    _exit();
  }

  /* now got enough data and you can add operations here… */

  _exit();
}
```

In the codes, we use two macros: DEFINE_AT and LE16_TO_CC. The former can define a variable at the specified address of the specified memory area while the use of the latter will be explained later in the topic of “Big-Endian and Little-Endian”, so you can temporarily think of it as useless. In this sample, we have defined a 512-byte array of unsigned char at 0x400 of the external memory.

Furthermore, we have also declared a structure where *offset* is designed to specify the location where the data input this time are stored in the variable *big_buff*. *len* indicates the length of the data input this time while *buff* saves the inputted data.

Suppose the 512-byte data shall be transported to Senselock ELbefore they are operated, the above codes can be executed four times. At each execution, the following offsets: `0, 128, 256, 384` and their corresponding 128-byte data are transmitted in respectively. At the fifth execution, the variable *`len`* is set to `0,` indicating no additional data can be transmitted in. At this point, the codes have acquired sufficient 512-byte data and can continue to execute the required functions.

## 3. Code Memory

Although Code Memory does not belong to the general memory structure we mentioned above, it can be operated (read only) in a similar way . Under some specific circumstances (for example, one may need to put a big consulting table in SenseLock EL), the use of Code Memory can provide great convenience.

The Code Memory can also be used to store some initial variables, such as strings, so as to save the memory usage. For example, for the codes below:

```
code char message[1024] = "This should be a long message…";
```

If the variable is stored at the code section, you can make read-only operation on the variable just like operating general memory in SenseLock EL.

Attention: None of the "system functions" can operate Code Memory, so to operate the variable defined in Code Memory via "system function", you must use functions like strcpy(), memcpy(), etc to copy them to the general memory area.

## 4. Alignment issue

To transport data between the computer and Senselock ELby copying the structures requires the attention for the alignment issue. In high-level languages, compiler will align the structures by default in accordance with the machine instruction so as to enhance operation efficiency. For example, if we use Visual C++ 6.0 to define the following structure:

```
struct {
  char c;
  long l;
} x;
```

The structure will take up 8-byte space. Whereas the same defined structure might take up 5-byte memory space in SenseLock EL. If data between the computer software and Senselock ELare transported via same structure, there might arise chaos. In fact, there exists such a problem in the sample of memory sharing mentioned above.

The solution requires that the structure defined in computer software should be aligned by 1 byte. For example, in Visual C++ 6.0[8], we can add the compiler directive `#pragma pack(push, 1)` before the structure declaration to force the compiler to align it by 1 byte. After the declaration is made, add `#pragma pack(pop)` to restore the original setting. For example,

```
#include "windows.h"

#pragma pack(push, 1)

typedef struct _MY_STRUCTURE {
  char c;
  long l;
} MY_STRUCTURE

#pragma pack(pop)
```

[8] There are similar forcing methods in other high-level languages. For example, the packed key word can be used in Delphi. If you do not want to change the setting of the compiler, you can also insert additional member variable in the definition of the structure to achieve the goal of alignment. For example, in the definition of the structure x, you can insert additional three bytes char padding[3] behind char c.

For internal codes of SenseLock EL, if the Raisonance Rkit compiler is used, it is necessary to do setup in the compilation option. For detail, please refer to Chapter 5 "Direction for Compiler Usage". The other two compilers do not need to be set.

### 5. Big-Endian and Little-Endian

This is an issue concerning the CPU architecture. Let us first explain briefly the meanings of the two terms:

Big-Endian indicates "high byte at the low address of the memory". In the computer of Big-Endian, the high byte of the variable is in the front;

Little-Endian indicates "low byte at the low address of the memory". In the computer of Little-Endian, the low byte of the variable is in the front.

For example, to define the variable `unsigned long x = 0x12345678`, in the computer of Big-Endian, the variable *x* in the memory shall be stored like this: `12 34 56 78` while in the computer of Little-Endian, the way of storage is just the opposite: `78 56 34 12`. An error will occur if a value is assigned to the variable on the computers of different types by means of memory copying.

The X86 serial computers are based on Little-Endian while Macintosh on Big-Endian. Keil C51 uses Big-Endian while other two kinds of compilers use Little-Endian. The communication buffer is a direct copy of the memory data, so this issue shall be taken into account.

Let's still take unsigned long x = 0x12345678 for example. Suppose the four bytes are transmitted to the codes compiled by Keil C51 via the communication buffer. At this point, the sequence of the bytes in the communication buffer is 78 56 34 12. If using the following code to assign y, the result of y will be 0x78563412, which obviously falls out of our expectation.

```
…
unsigned long y;
memcpy(&y, pbInBuff, sizeof(unsigned long));
…
```

The solution to this is to convert the byte sequence. When the data transported via the communication buffer contain multi-byte variables, such as short, long, float and other types, conversion must be done. The conversion can either be conducted in the computer software or in Senselock ELEXF. To achieve better code compatibility, it is recommended to conduct conversion in Senselock ELEXF.

Some pre-defined macros can assist in the conversion of byte sequence. These macros can determine automatically whether there is a need for conversion depending on the compiler type without the necessity of your considering what compiler to use. For example, LE16_TO_CC can convert 16 bit variable from Little-Endian into the byte sequence used in SenseLock EL. Other macros accomplish the similar functions. Abbreviation BE indicates Big-Endian while CC indicates Senselock ELcompiler (C Compiler).

Attention: As the double-precision floating point is not the standard data type supported by the compiler, but a structure (DOUBLE_T) self-defined by Senselock, this data type has no bearing to the compiler. No matter what compiler is used, DOUBLE_T is always of Little-Endian.

## 4.1.4. Use System Function Library Files

This section does not discuss the details of system functions. Rather, it examines how to use the library files of the system function in the compilation of Senselock ELcodes.

If the Keil C51 compiler is used, there are library files in three modes available: *ses51c.lib*, *ses51s.lib* and *ses51l.lib*, which correspond respectively to Compact Model, Small Model and Large Model in compilation mode. We do not recommend the two modes other than Large Model. To compile the codes, please copy the corresponding library files to the directory where the project is located and add them to the project.

It is somewhat simpler to use Raisonance Rkit. There is only one library file *ses.lib*, so simply copying it to the project will do the job. However, it shall be noted that Rkit has a requirement for the sequence in which files are added. *ses.lib* must be the last file added to the project.

### 4.1.5. Data Type

The description of part of data types supported by Senselock ELis given below:

| Type | `char` | `int` | `short` | `long` | `float` | `double` |
|---|---|---|---|---|---|---|
| Length (byte) | 1 | 2 | 2 | 4 | 4 | 8 |

### 4.1.6. Code Compilation Proposal

Given below are some general proposals for compiling Senselock ELinternal codes:

1. Try as much as possible to use global variables to avoid stack overflow error as a result of too many local variables, especially arrays and structures;

2. Try not to use such variables as arrays, structures, etc. that might occupy a large amount of memory directly when passing parameters. Instead, pointers shall be used to avoid possible stack overflow;

3. There should not be too many levels of function nesting so as to avoid possible stack overflow;

4. Under the VM mode, try as much as possible to use external memory to avoid memory overflow or stack insufficiency;

5. Consider the issue of alignment to avoid data communication errors;

6. Consider the issue of Big-Endian and Little-Endian to avoid data communication errors;

7. Try as much as possible to use the system function to enhance efficiency. For example, `_mem_copy()` is higher than `memcpy()` in efficiency;

8. Use English comments as some compilers do not support Chinese currently(i.e. RKit);

9. Try as much as possible not to operate strings with functions like `sprintf(), sscanf()`, etc. to avoid possible stack overflow;

## 4.2. Access Senselock ELDevice

In the software, the Senselock ELdevice is accessed via API offered by us. There are two levels of access: developer level and user level. Both of them can get their corresponding access authorization using their respective PIN. This section will exemplify the general process of accessing the Senselock ELdevice in the software.

All the samples in this section are developed and tested in Visual C++ 6.0.

### 4.2.1. General Access

General access refers to access that does not require security authorization, which involves

Listing/searching and connecting device, switching directory, sending control code, closing device, etc. Please refer to the sample below:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "sense4.h"

int main(int argc, char** argv)
{
  SENSE4_CONTEXT s4ctx = {0}; /* current device context */
  SENSE4_CONTEXT *ps4ctx = NULL; /* for device context list */
  unsigned long ctx_size = 0; /* size of device context list */
  unsigned long ret = 0; /* return value */
  unsigned long len = 0; /* for returned data length. */

  /*step 1: Enumerate all the Senselock ELdevices connected. */
  ret = S4Enum(NULL, &ctx_size);
  if (ret != S4_SUCCESS && ret != S4_INSUFICIENT_BUFFER)
  {
    printf("Enumerate Senselock ELfailed!<error code = %08x>\n", ret);
    return 1;
  }

  if (ctx_size == 0)
  {
    printf("No Senselock ELfound!\n");
    return 1;
  }

  ps4ctx = (SENSE4_CONTEXT *)malloc(ctx_size);
  ret = S4Enum(ps4ctx, &ctx_size);
  if (ret != S4_SUCCESS) /* Some error occured. */
  {
    printf("Enumerate Senselock ELfailed!<error code = %08x>\n", ret);
    free(ps4ctx);
    ps4ctx = NULL;
    return 1;
  }

  /* step 2: Open one of the device. For example, the first one. */
  memcpy(&s4ctx, ps4ctx, sizeof(SENSE4_CONTEXT));
  /* Here we can get device's Global Serial Number via s4ctx.bID. */

  free(ps4ctx); /* Or free it later if you want to access other devices.*/
  ps4ctx = NULL;

  ret = S4Open(&s4ctx);
  if (ret != S4_SUCCESS)
  {
    printf("Open first Senselock EL failed!<error code = %08x>\n", ret);
    return 1;
  }

  /* step 3: Now we can send some control code to device,
    for example, control the LED. This step is not necessary. */
  ret = S4Control(&s4ctx, S4_LED_UP, NULL, 0, NULL, 0, &len);
  if (ret != S4_SUCCESS)
  {
    printf("Light LED failed!<error code = %08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  /* step 4: We may change the current directory here. */
  ret = S4ChangeDir(&s4ctx, "\\"); /* Change to root dir. */
  if (ret != S4_SUCCESS)
  {
    printf("Change to root dir failed!<error code = %08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  /* step 5: Do some more actions here... */

  /* step 6: Close device. */
  S4Close(&s4ctx);


  return 0;
}
```

This sample program describes the general process of accessing Senselock EL in 6 steps:

| | | |
|---|---|---|
| 1. | List/search Senselock EL devices | It is necessary to call `S4Enum()` function twice in all. At the first time, set the first parameter to NULL and the value of the second parameter to 0. After the first call, the function will list all the Senselock EL devices in the system and the desired sizes of device context (the amount of the devices can be calculated this way: `ctx_size/sizeof(SENSE4_CONTEXT)`.<br><br>Allocate dynamically the desired memory so as to save the context list of all the devices listed(or allocate a block of sufficiently large static memory), call the `S4Enum()` function again and it will copy the context of all the devices to the list.<br><br>The context of each device contains the serial number[9] of the device. Under the situation where there are several devices, you can differentiate devices with serial numbers. |
| 2. | Open specified device | The device can be opened by passing in the context pointer of the specified device as a parameter to `S4Open()` function. |
| 3. | Send control code | After opening the device, you can control the device by sending the control codes, such as modifying LED status, modifying communication mode, getting hardware serial number, etc. |
| 4. | Change current directory | When you access the device for the first time, the current directory is the root directory. To change the current directory to sub-directory requires the calling of `S4ChangeDir()` function or the calling of this function to switch from sub-directory to root directory. |
| 5. | Other operation types | Other security level related accesses such as PIN authentication, file writing, program calling, etc. |
| 6. | Close device | After all accesses are completed, close Senselock EL devices. |

### 4.2.2. Developer level Access

Generally speaking, developer level access can be made via the tool software we have offered. If you hope to manage your own device or to improve code test efficiency, you can complete operations via developer level API function.

Developer level access involves creating directory (root directory and sub-directory), deleting (clearing) directory, creating a file, overwriting a file and modifying developer level PIN.

[9] Senselock EL offers a global unique hardware serial number, but if a user-defined ATR file is created under the root directory, the serial number in the context will be the user-defined serial number. In this case, the real hardware serial number can be only acquired via `S4Control()`. For the details of “User Defined Serial Number”, please refer to the description of `S4CreateDir()` function.

To ease code interpretation, we first encapsulate the device connection part of sample code in 4.2.1 into a function: `OpenSense4()`, which opens the device of specified *`index`* and returns the context of the device. The source codes are as follows:

```
unsigned long OpenSense4(SENSE4_CONTEXT *ctx, int index)
{
  SENSE4_CONTEXT *ps4ctx = NULL;
  unsigned long ctx_size = 0;
  unsigned long ret = 0;

  If (ctx == NULL)
  {
    return S4_INVALID_PARAMETER;
  }

  /* Enumerate all the Senselock ELdevices connected. */
  ret = S4Enum(NULL, &ctx_size);
  if (ret != S4_SUCCESS && ret != S4_INSUFFICIENT_BUFFER)
  {
    return ret;
  }

  if (ctx_size == 0)
  {
    return S4_KEY_REMOVED;
  }

  ps4ctx = (SENSE4_CONTEXT *)malloc(ctx_size);

  ret = S4Enum(ps4ctx, &ctx_size);
  if (ret != S4_SUCCESS) // Some error occured.
  {
    free(ps4ctx);
    ps4ctx = NULL;
    return ret;
  }

  if (index > (int)(ctx_size/sizeof(SENSE4_CONTEXT) - 1))
  {
    return S4_KEY_REMOVED;
  }

  memcpy(ctx, ps4ctx+index, sizeof(SENSE4_CONTEXT));
  free(ps4ctx);
  ps4ctx = NULL;

  ret = S4Open(ctx);

  return ret;
}
```

The sample codes given below demonstrate how to re-initialize a Senselock EL device, create and write an executable file and a data file, modify the developer level PIN of Senselock EL root directory. This is also a general process of initializing Senselock EL device.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "sense4.h"
#include "psense4.h"

int main(int argc, char **argv)
{
  SENSE4_CONTEXT s4ctx = {0};
  unsigned char fid_exe[] = "d001"; // exe file ID
  unsigned long exe_size = 2048; // create file size
  unsigned char fid_dat[] = "d002"; // data file ID
  unsigned long dat_size = 1024; // create file size
```

```
char exe_path[] = "c:\\s4demo\\demo1.hex"; // exf file path in disk
char dat_path[] = "c:\\s4demo\\data1.dat"; // data file path in disk
unsigned char default_dev_pin[] = "123456781234567812345678";
unsigned char old_dev_pin[] = "123456781234567812345678";
unsigned char new_dev_pin[] = "876543218765432187654321";
unsigned long len = 0;
unsigned ret = 0;

/* Open first Senselock ELif exists. */
ret = OpenSense4(&s4ctx, 0);
if (ret != S4_SUCCESS)
{
  printf("Open Senselock ELfailed! <error code = 0x%08x>\n", ret);
  return 1;
}

/* Check whether a root dir exists. */
ret = S4ChangeDir(&s4ctx, "\\");
if (ret != S4_FILE_NOT_FOUND && ret != S4_SUCCESS)
{
  printf("Change to root dir failed! <error code = 0x%08x>\n", ret);
  S4Close(&s4ctx);
  return 1;
}

/* If a root dir exists. */
if (ret != S4_FILE_NOT_FOUND)
{
  /* Verify developer PIN to get full access privilege. */
  ret = S4VerifyPin(&s4ctx, old_dev_pin, 24, S4_DEV_PIN);
  if (ret != S4_SUCCESS)
  {
    printf("Verify dev PIN failed! <error code = 0x%08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  /* Delete old root dir. */
  ret = S4EraseDir(&s4ctx, NULL);
  if (ret != S4_SUCCESS)
  {
    printf("Delete root dir failed! <error code = 0x%08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }
}

/* Create new root dir. */
ret = S4CreateDir(&s4ctx, "\\", 0, S4_CREATE_ROOT_DIR);
if (ret !=  S4_SUCCESS)
{
  printf("Create new root failed! <error code = 0x%08x>\n", ret);
  S4Close(&s4ctx);
  return 1;
}

/* Verify developer PIN to get full access privilege. */
ret = S4VerifyPin(&s4ctx, default_dev_pin, 24, S4_DEV_PIN);
if (ret != S4_SUCCESS)
{
  printf("Verify dev PIN failed! <error code = 0x%08x>\n", ret);
  S4Close(&s4ctx);
  return 1;
}

/* Write disk file to SenseLock EL. */
ret = PS4WriteFile(&s4ctx,
  fid_exe, exe_path, &exe_size,
    S4_CREATE_NEW, S4_HEX_FILE,
    &len);
  if (ret != S4_SUCCESS)
  {
    printf("Write exe file failed! <error code = 0x%08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  ret = PS4WriteFile(&s4ctx,
    fid_dat, dat_path, &dat_size,
    S4_CREATE_NEW, S4_DATA_FILE,
    &len);
  if (ret != S4_SUCCESS)
  {
    printf("Write data file failed! <error code = 0x%08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  /* Change developer PIN. */
  ret = S4ChangePin(&s4ctx,
        default_dev_pin, 24,
        new_dev_pin, 24,
        S4_DEV_PIN);
  if (ret != S4_SUCCESS)
  {
    printf("Change dev PIN failed! <error code = 0x%08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  /* Close SenseLock EL. */
  S4Close(&s4ctx);

  /* everything done! */
  printf("Congratulations!\n");

  return 0;
}
```

The process of developer level access is relatively simple. The basic process is like this: open device→switch directory→authenticate developer PIN→developer level operation→close device. As the codes of developer level access might contain the “plain text” of developer PIN, make sure to keep confidentiality of the developer level access codes.

For the detailed description of API involved in the codes, please refer to Chapter 9 “API Reference”.

Important reminder: Under whatever condition, do not include developer level access codes in the software to be released. Please change the developer level PIN into a secret value before the dongle is released.

### 4.2.3. User Level Access

Except for the basic operations, there is actually only one access in user level: executing internal EXF.

In the sample codes as given below, we continue to use the `OpenSense4()` function as mentioned above.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "sense4.h"
```

```
#include "psense4.h"

int main(int argc, char **argv)
{
  SENSE4_CONTEXT s4ctx = {0};
  unsigned char fid_exe[] = "d001"; // exe file ID
  unsigned char user_pin[] = "12345678";
  unsigned long len = 0;
  unsigned ret = 0;
  unsigned char input[256] = {0};
  unsigned long input_len = 128;
  unsigned char output[256] = {0};
  unsigned long output_len = 251;

  /* Open first Senselock ELif exists. */
  ret = OpenSense4(&s4ctx, 0);
  if (ret != S4_SUCCESS)
  {
    printf("Open Senselock ELfailed! <error code = 0x%08x>\n", ret);
    return 1;
  }

  /* Change to root dir if exists. */
  ret = S4ChangeDir(&s4ctx, "\\");
  if (ret != S4_FILE_NOT_FOUND && ret != S4_SUCCESS)
  {
    printf("Change to root dir failed! <error code = 0x%08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  /* Verify user PIN to get invoking privilege. */
  ret = S4VerifyPin(&s4ctx, user_pin, 8, S4_USER_PIN);
  if (ret != S4_SUCCESS)
  {
    printf("Verify user PIN failed! <error code = 0x%08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  /* Invoke exf 0xd001. */
  ret = S4Execute(&s4ctx, "d001",
         input, input_len, output, output_len, &len);

  if (ret != S4_SUCCESS)
  {
    printf("Invoke 0xd001 failed! <error code = 0x%08x>\n", ret);
    S4Close(&s4ctx);
    return 1;
  }

  /* Close SenseLock EL. */
  S4Close(&s4ctx);

  /* everything done! */
  printf("Congratulations!\n");

  return 0;
}
```

If you hope to use XA running mode, it is necessary to use the newer function `S4ExecuteEx()`. If you hope the software accesses the dongle in an exclusive mode, please replace `S4Open()` with the `S4OpenEx()` function. For details, see Chapter 9 “API Reference”.

Important reminder: In actual application, try mixing or encrypting communication data so as to get higher randomness to thwart malicious decryption analysis.

## 4.3. About Code Test

From the perspective of Senselock EL internal codes, you can conduct tests in two ways: one is hardware level test, namely, downloading compiled codes directly to Senselock EL device and then calling and checking running results; the other is software emulation test. With the simulator we have offered, code test can be made. The advantage of the latter is that it can also trace and debug codes in addition to tests. The disadvantage is that some extra configuration is required.

From the perspective of device call, you can conduct tests in two ways: one is to use the tool software “development test tool” offered by Senselock; the other is to write high-level language testing codes .

Which testing method to choose depends on your own habits. The process of software emulation test has bearing to the compiler. For detailed usage, see Chapter 5 “Direction for Compiler Usage”. For tool software, see Chapter 6 “Direction for Tool Software Usage”.

# 5. Direction for Compiler Usage

There are now three kinds of compiler software available for the development of Senselock ELEXFs. This chapter will give a description of the usage and precautions for these compilers. If you hope to understand the usage of compiler software in a more visual way, please refer to "Multimedia Teaching System" in the SDK CD.

For the installation method of Keil and Rainsonance compiler, please refer to relevant instruction documents of them.

Senselock EL2.3 supports two running modes: VM and XA. The VM251 mode supported by the version before 2.2 is already replaced by the XA mode. To achieve a fast understanding of the three kinds of compiler, please refer to Table 5-1 first.

Table 5-1 Features of Compiler Software

| | Sense Skit | Keil C51 | Rkit |
|---|---|---|---|
| Code mode | VM | | XA |
| Code bit | 8 bits | | 16 bits |
| Byte sequence | LE | BE | LE |
| Code optimization efficiency | middle | high | High |
| Memory structure | Internal RAM+external RAM | | No distinction between internal and external |
| Other descriptions | Free software, full functions, but relatively low in efficiency | Commercial software. Special attention shall be given to the issue of byte sequence. | Commercial software. We have not yet offered the debugging simulator at this stage. |

LE: Little-Endian BE: Big-Endian

As standard C programming is used, whatever compilation software is used, it will only have some impact on development habits, but no much on the codes themselves. By using the macros defined in *ses_v3.h* effectively, you can compose compiler-independent codes.

Attention: The compilation result of the compiler is HEX file, which is in text format, so it cannot be run directly in SenseLock EL. It must be converted into binary codes. The Hexbin tool available in SDK can be used in the middle of conversion. For details, please refer to Section 5.4 "Executable File Format".

## 5.1. Direction for Keil C51 Usage

Keil C51 is a suite of excellent compilation software developed by Keil. The IDE of the software is called µVision. You can download the beta version from the official website of Keil: *http://www.keil.com*. For limitation in the beta version, please refer to the instruction on the website.

This section gives a necessary description of Keil C51 usage. To achieve a better understanding of Keil C51 compilation software, you are recommended to refer to the help document of the software.

All the three kinds of compilation software introduced in this chapter manage codes by "project". For Keil C51, a typical project usually includes project file **.uv2*, several C code files (including header files), Senselock EL system function header file *ses_v3.h* and Senselock ELsystem function library file *ses51?.lib*[10].

If you use dongle 2.2 or below and hope to use float algorithm, you must overwrite the six

[10] '?' indicates S, C or L, corresponding to three compilation modes: Small, Compact and Large.

files under the directory of *%Keil%\C51\Lib* with the six files with identical names under the directory of *%SDK%\IDE\Keil\FloatLib*[11]; otherwise an error will occur in the float calculation. To be on the safe side, it is recommended to back up the six files before overwriting them.

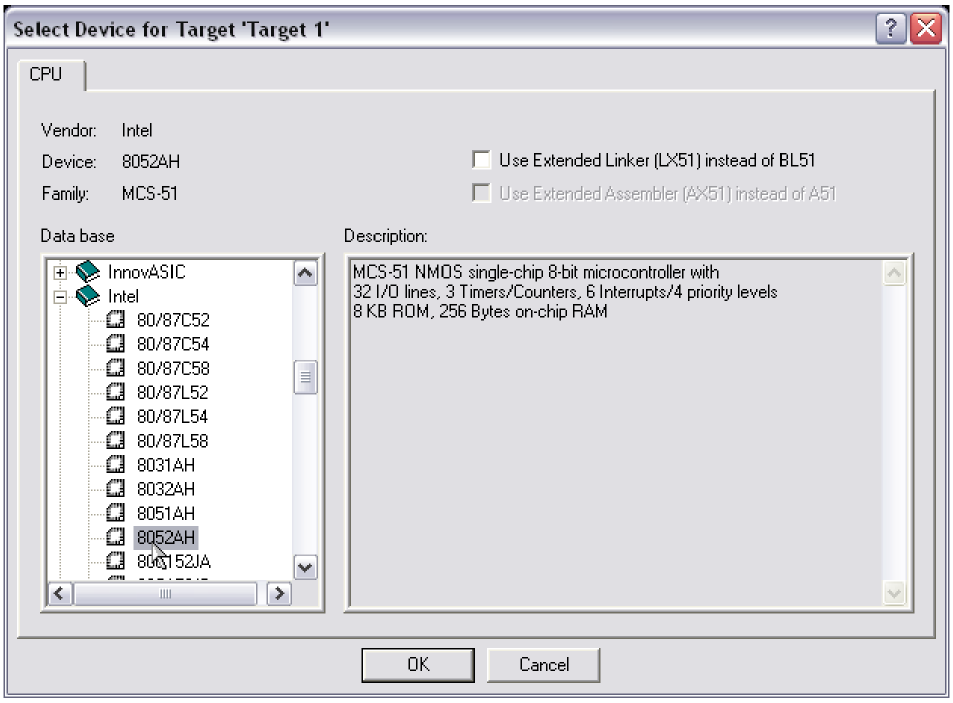
To make it easy to configure the Keil compilation environment, a patch installation program (KeilPatch.exe in the directory of *%SDK%\IDE\KEIL*） is available in SDK. This patch can be used to fulfill automatically such functions as library file copying, debugger setup, etc. For details, please refer to the instruction file in the patch directory.

### 5.1.1. Project Creation and Management

The process of creating a project goes like this (there might be some differences in operation processes if µVision of a different version is used. In this text, we will take µVision 3 for example）:

1. Start the software, choose the menu “Project”→“New Project…” and save the project file to the disk;

2. Choose “Intel”→“8052AH” from the “Data Base” list box of the “Select Device for Target ‘Target1’” dialog box. For the existing items, you can click “Select Device for Target ‘Target1’” from the “Project” menu, open “Select Device for Target ‘Target1’” dialog box and choose “Intel”—>“8052AH”CPU from the “Data Base” list box as shown in Figure 5-1.


Select Device for Target 'Target 1'

CPU

Vendor: Intel
Device: 8052AH
Family: MCS-51

[ ] Use Extended Linker (LX51) instead of BL51
[ ] Use Extended Assembler (AX51) instead of A51

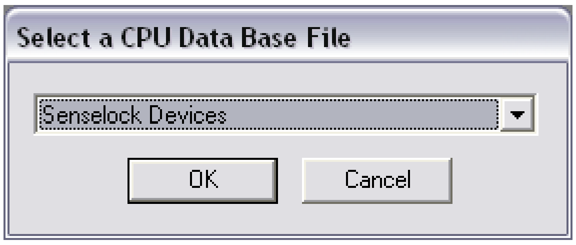
Data base: InnovASIC, Intel, 80/87C52, 80/87C54, 80/87C58, 80/87L52, 80/87L54, 80/87L58, 8031AH, 8032AH, 8051AH, 8052AH, 80C152JA

Description:
MCS-51 NMOS single-chip 8-bit microcontroller with
32 I/O lines, 3 Timers/Counters, 6 Interrupts/4 priority levels
8 KB ROM, 256 Bytes on-chip RAM

Figure 5-1 CPU Type Selection

If the patch installation program is already used to configure the Keil development environment, there might be slight differences in the above operation steps. You will first be prompted to choose the device database file as shown in Figure 5-2:

[11] *%SDK%* indicates the installation path for Senselock ELSDK. *%Keil%* indicates the installation path for Keil C51. *%RIDE%* indicates the installation path for Rkit. Similarly hereinafter.

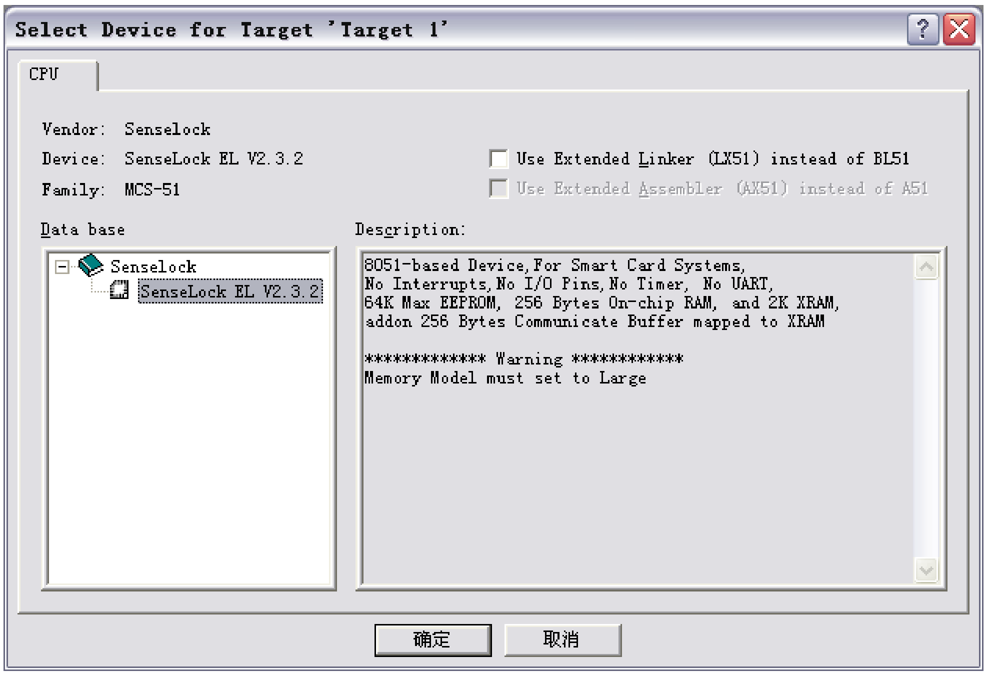


Select a CPU Data Base File

Senselock Devices

Figure 5-2 Select a Device Database File

Choose Senselock Devices and there will show up a dialog box of "Select Device for Target 'Target1'". The "Description" field displays some basic information of the hardware and prompts you to choose Memory Model in "Large" mode as shown in Figure 5-3:



Select Device for Target 'Target 1'

Vendor: Senselock
Device: SenseLock EL V2.3.2
Family: MCS-51

☐ Use Extended Linker (LX51) instead of BL51
☐ Use Extended Assembler (AX51) instead of A51

Data base: Senselock / SenseLock EL V2.3.2

Description:

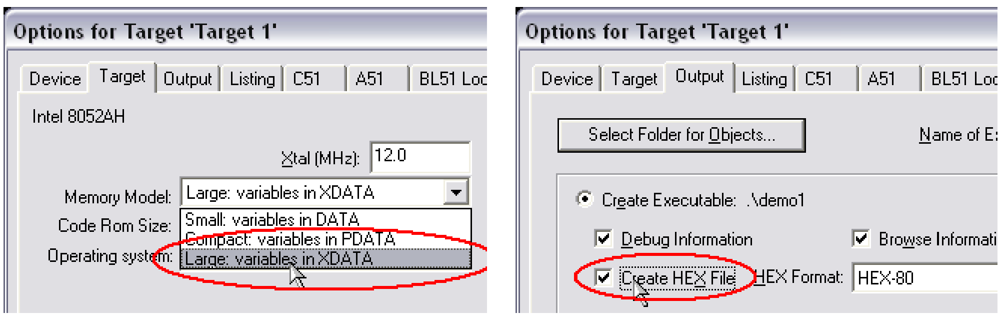
```
8051-based Device,For Smart Card Systems,
No Interrupts,No I/O Pins,No Timer, No UART,
64K Max EEPROM, 256 Bytes On-chip RAM, and 2K XRAM,
addon 256 Bytes Communicate Buffer mapped to XRAM

************* Warning ************
Memory Model must set to Large
```

Figure 5-3 Choose Device Type

From here, choose "Senselock"→"Senselock ELV2.3.2"

3. Set project options. Click the menu "Project"→"Options for Target 'Target 1'", and choose "Memory Model" as "Large" in the "Target" page as shown in Figure 5-4;



Options for Target 'Target 1'

Intel 8052AH

Xtal (MHz): 12.0

Memory Model: Large: variables in XDATA

Code Rom Size:

Operating system:

Small: variables in DATA
Compact: variables in PDATA
Large: variables in XDATA

Options for Target 'Target 1'

Create Executable: .\demo1

☑ Debug Information ☑ Browse Informati

☑ Create HEX File HEX Format: HEX-80

Figure 5-4 Project Option Configuration

4. Continue to set project option. In the "Output" page, choose the checkbox "Create HEX

File" and only this way, the target file in hex format can be generated in project compilation. Close the project option dialog box;

5. Make sure *ses_v3.h* is already copied to the current project directory or to the directory of "*%keil%\C51\INC";* otherwise there will occur an error of failure to find the header file *ses_v3.h* in the middle of program compilation. *Ses_v3.h* is located under the directory of *"%SDK%\IDE\KEIL\INC;*

6. Depending on which "Memory Model" is selected in step 3, add the corresponding SES library file to the project. For example, if "Memory Model" is chosen as Large, add ses51L.lib to the project. This file is located in the directory of "*%SDK%\IDE\KEIL\LIB"*. Choose the menu "Project"→"Components, Environment, Books…" to add the file to the project;

7. Add the C source codes written by yourself to the project;

After the project is successfully created, you can modify the options of the project at any time.

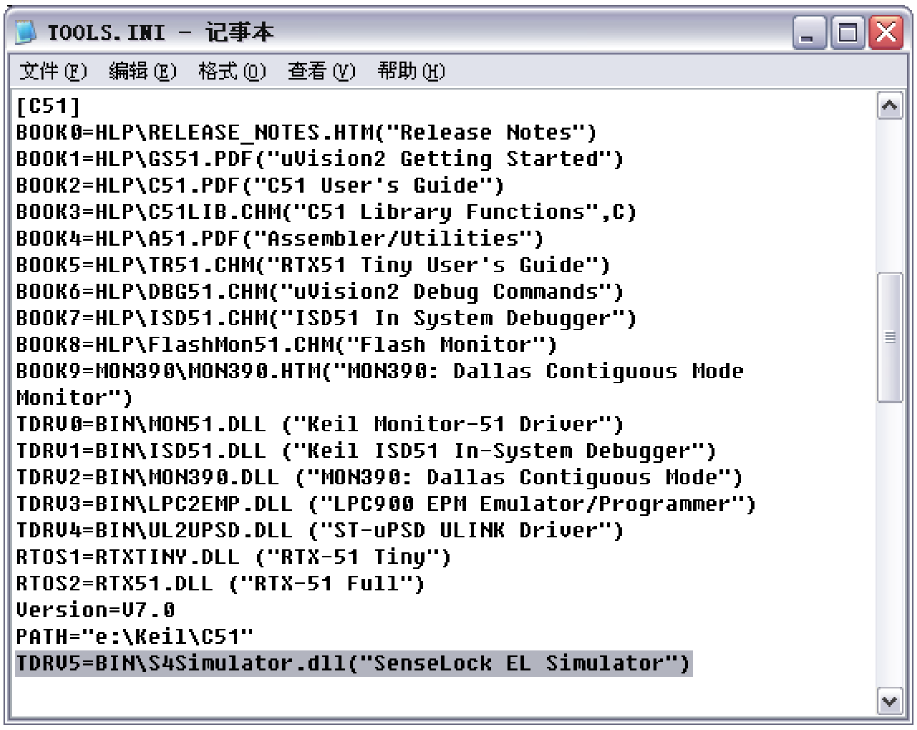
### 5.1.2. Code Compilation and Debugging Configuration

There is nothing special in the middle of compilation. Simply press F7 key or choose the concerned menu to get started with compilation. If the compilation is successful, a HEX file will be generated accordingly in the project directory; otherwise the compiler will give a prompt for error.

You can trace and debug codes using the software simulator offered by Senselock in Keil C51. To use the simulator, you must modify the configuration of the compiler software. You can use the patch installation tool in the directory of *`%SDK%\IDE\KEIL`* to complete configuration automatically or do it manually in this way:

1. Copy *S4Simulator.dll* and *vfsView.exe* shipped with SDK in the directory of *%SDK%\IDE\KEIL\bin\* to the directory of *%KEIL%\c51\bin*;

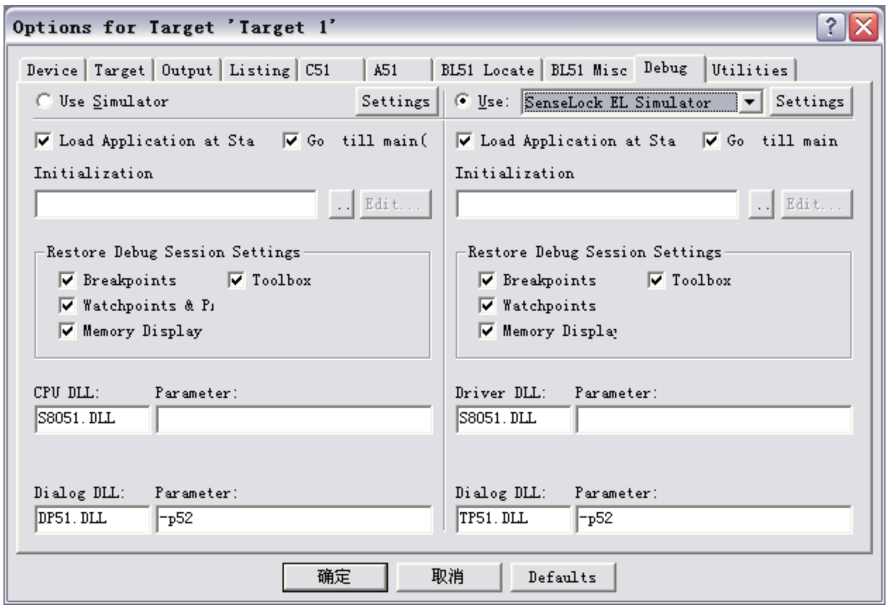
2. Open the configuration file *%KEIL%\Tools.ini* and add the following to the `[C51]` section of the file: TDRV0=BIN\S4Simulator.dll("Senselock ELSimulator"). If TDRV0 is already used, use the next available number TDRV1, and so on , as shown in Figure 5-5.



```
[C51]
BOOK0=HLP\RELEASE_NOTES.HTM("Release Notes")
BOOK1=HLP\GS51.PDF("uVision2 Getting Started")
BOOK2=HLP\C51.PDF("C51 User's Guide")
BOOK3=HLP\C51LIB.CHM("C51 Library Functions",C)
BOOK4=HLP\A51.PDF("Assembler/Utilities")
BOOK5=HLP\TR51.CHM("RTX51 Tiny User's Guide")
BOOK6=HLP\DBG51.CHM("uVision2 Debug Commands")
BOOK7=HLP\ISD51.CHM("ISD51 In System Debugger")
BOOK8=HLP\FlashMon51.CHM("Flash Monitor")
BOOK9=MON390\MON390.HTM("MON390: Dallas Contiguous Mode
Monitor")
TDRV0=BIN\MON51.DLL ("Keil Monitor-51 Driver")
TDRV1=BIN\ISD51.DLL ("Keil ISD51 In-System Debugger")
TDRV2=BIN\MON390.DLL ("MON390: Dallas Contiguous Mode")
TDRV3=BIN\LPC2EMP.DLL ("LPC900 EPM Emulator/Programmer")
TDRV4=BIN\UL2UPSD.DLL ("ST-uPSD ULINK Driver")
RTOS1=RTXTINY.DLL ("RTX-51 Tiny")
RTOS2=RTX51.DLL ("RTX-51 Full")
Version=V7.0
PATH="e:\Keil\C51"
TDRV5=BIN\S4Simulator.dll("SenseLock EL Simulator")
```

Figure 5-5 Keil C51 Simulator Configuration

Start Keil μVision to open the debugging project, choose "Use Senselock ELSimulator" on the right of the "Debug" page in project option, and configure debug options according to Figure 5-6:


Options for Target 'Target 1'
Device | Target | Output | Listing | C51 | A51 | BL51 Locate | BL51 Misc | Debug | Utilities
Settings
Settings
Edit...
Edit...
确定
取消
Defaults

Use Simulator ○ | Use: ● SenseLock EL Simulator

Load Application at Sta ☑ Go till main( ☑ | Load Application at Sta ☑ Go till main ☑

Initialization: | Initialization:

Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watchpoints & P ☑ Memory Display | Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watchpoints ☑ Memory Display

CPU DLL: S8051.DLL Parameter: | Driver DLL: S8051.DLL Parameter:

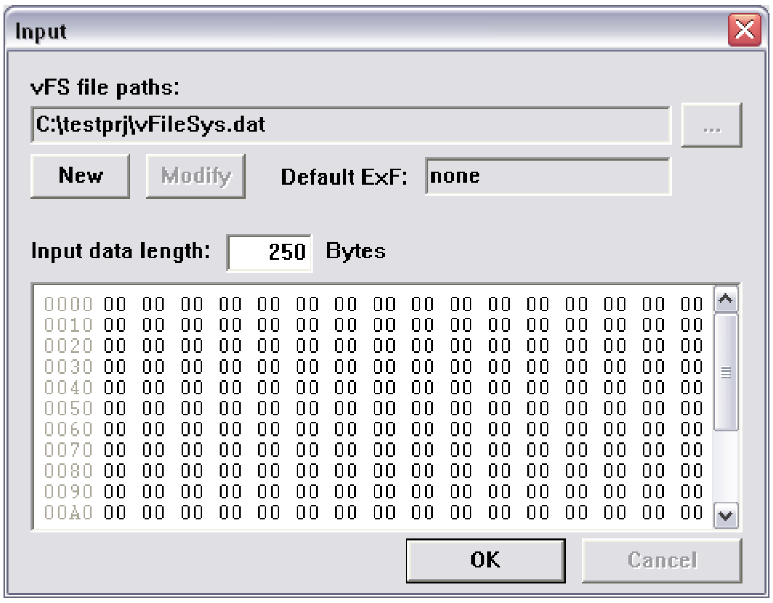
Dialog DLL: DP51.DLL Parameter: -p52 | Dialog DLL: TP51.DLL Parameter: -p52

Figure 5-6 Debug Option Configuration

Close the project option dialog box. Now you can debug the codes in the project.

### 5.1.3. Code Debugging Method

The way Keil enters the debug status is to choose the menu "Debug"→ "Start/Stop Debug Session" or click the icon on the tool bar. If the Senselock ELsoftware simulator is properly installed and configured, when Keil enters the debugging status, there will be the following data input window:



vFS file paths: C:\testprj\vFileSys.dat
Default ExF: none
Input data length: 250 Bytes

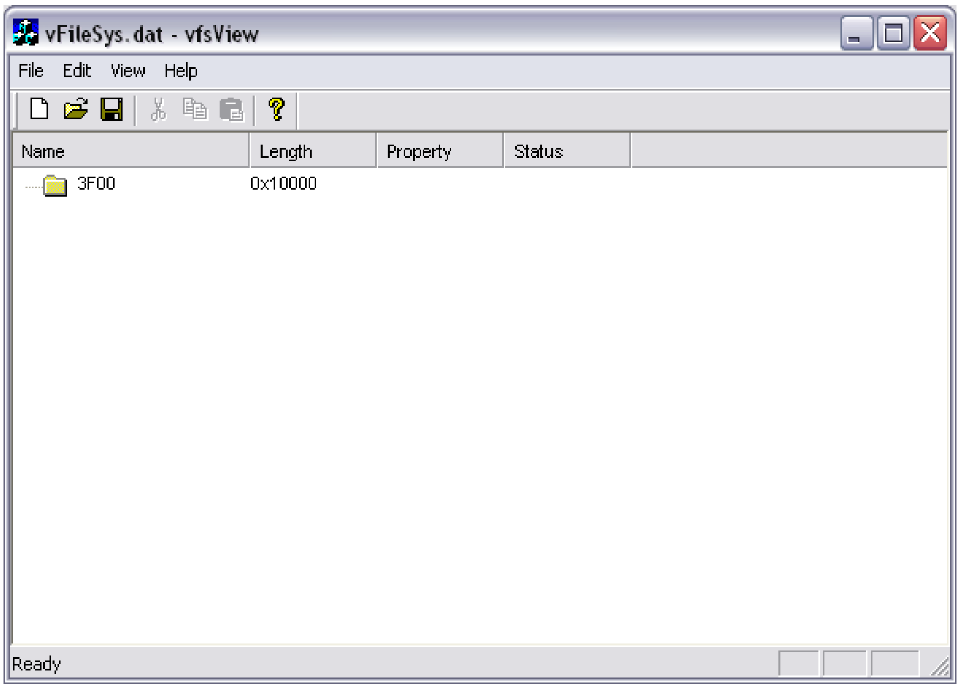
```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 5-7 Data Input Window

There are two parts of data input window. The upper half manages the Senselock ELsimulation file system while the lower half sets the data to be transported to Senselock ELcommunication buffer.

Senselock ELsimulator simulates Senselock ELfile system using the file stored in the current project directory called “vFileSys.dat”. Click the “New” button in the data input window to create a new simulation file system. It is not necessary to create a simulation file system for Senselock ELapplications that do not call the file system service.

Click the “Modify” button in the data input window to modify the simulation file system.

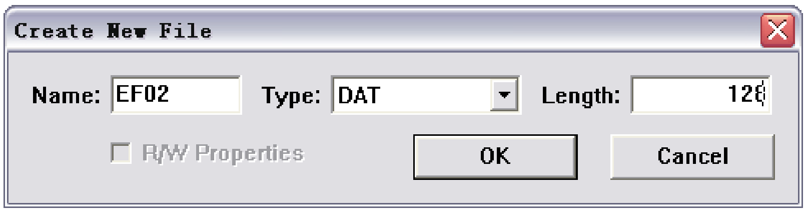


| Name | Length | Property | Status |
|---|---|---|---|
| 3F00 | 0x10000 | | |

Figure 5-8 Simulation File System Tool Interface

In the simulation file system, the file system takes the directory whose ID is "3F00" (namely, MF of SenseLock EL) as a root node and display the information by a tree structure. Designed only for code debugging, the simulation file system supports only one-level directory structure. No sub-directory can be created.

Choose "New File" from the right-button menu of the directory node to create a new file in the current directory.

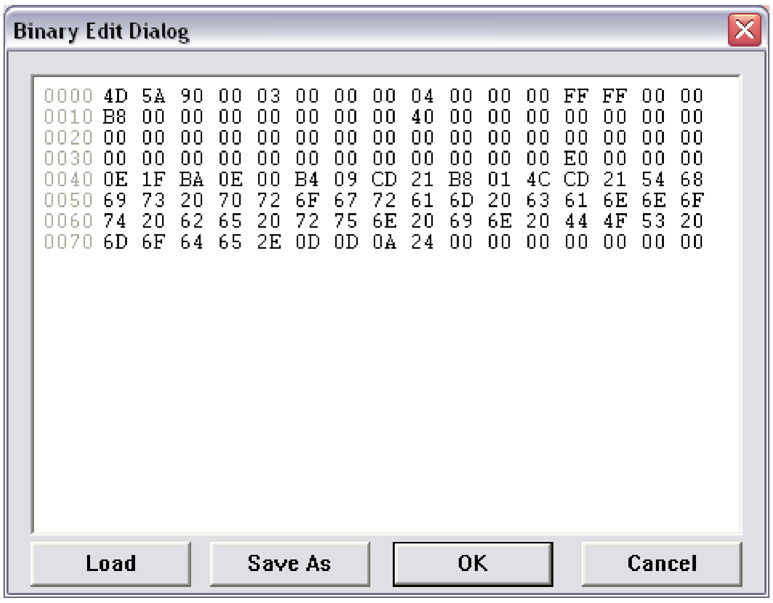


Name: EF02  Type: DAT  Length: 128

Figure 5-9 "Create a New File" Window

The simulation file system supports four types of files: executable files (EXF), data files (DAT), RSA public key files and private key files. For the public key file type, the file shall not be less than 136 bytes while for the private key file type, the file shall not be less than 330 bytes. The internal readable/writable files shall be marked with "R/W" attributes.

Double click or choose "Edit" from the right-button menu of any file node to edit its contents as shown in Figure 5-10.



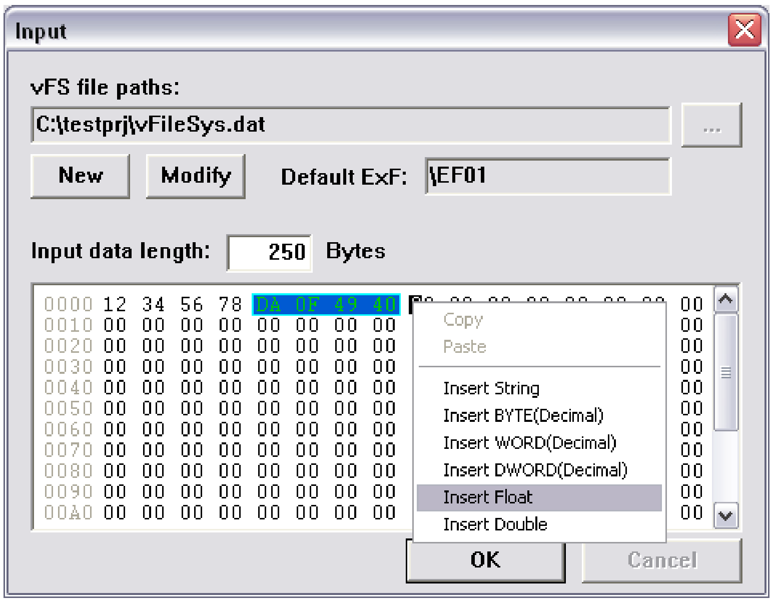
```
0000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
0010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00
0040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
0050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F
0060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
0070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00
```

Figure 5-10 Edit File Contents

You can modify the hexadecimal data directly when editing a file or choose to import data from a disk file or save the data to a disk file.

Sometimes executable files need to operate themselves. For example, the internal function `_create()` can be designed to acquire the information of the current executable file. As the current executable file is not saved to the simulation file system in a real sense, but only debugged in the Keil compilation environment, this kind of operation will fail. The solution is to mark an executable file as "default file" in the simulation file system (set this by right clicking the file node). In this way, when the executable file needs to operate itself, it will operate the "default file" as the current executable file.

Like the real hardware device, the software simulator simulates the communication buffer so as to exchange data with the outside. The data in the input data edit field will be copied to the communication buffer inside the software simulator for use by Senselock ELprograms. The data edit field allows for the input of hexadecimal data or formatted data. Right click an address to choose to input different data on the menu as shown in Figure 5-11.

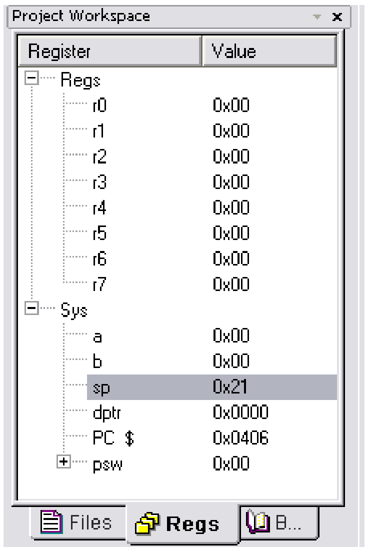


```
0000 12 34 56 78 DA 0F 49 40
0010 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00
```

Figure 5-11 Data Input Demo

After setting the file system and the data to be transported to SenseLock EL, click the “OK” button in the dialog box to enter the Keil debug status. In the debug status, the process of code debugging is more or less the same as other IDE, so no more description will be given here. Should you have any doubt, please refer to the instruction manual of Keil or visit http://www.keil.com for help.

If there shows up a dialog box for Senselock ELexecution error in the middle of debugging, in addition to viewing error codes, you might also check the Regs window on the left of Workspace area:



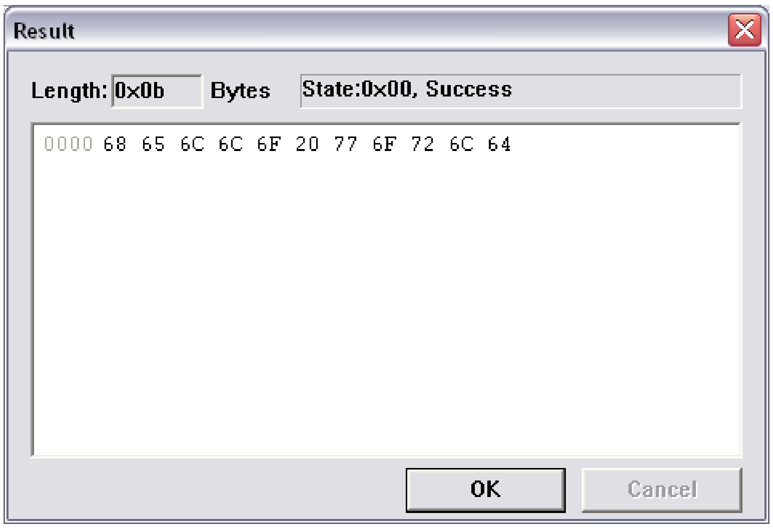
| Register | Value |
|---|---|
| Regs | |
| r0 | 0x00 |
| r1 | 0x00 |
| r2 | 0x00 |
| r3 | 0x00 |
| r4 | 0x00 |
| r5 | 0x00 |
| r6 | 0x00 |
| r7 | 0x00 |
| Sys | |
| a | 0x00 |
| b | 0x00 |
| sp | 0x21 |
| dptr | 0x0000 |
| PC $ | 0x0406 |
| psw | 0x00 |

Figure 5-12 Register Window

The Regs window displays the current CPU register list: R0~R7 are universal registers and R0~R1 are also indirect addressing registers of internal memory (IRAM); dptr is the indirect addressing register of external memory (XRAM); sp is the stack pointer; PC is the program pointer. The most likely errors include dptr out of range (the normal value is between 0x0000~0x08ff), SP overflow (the normal value is between 0x07~0xff, moving upward) or PC out of range (exceeding the valid code range).

The way of exiting from the debugging status is the same as that of entering debugging status. Choose menu "Debug"→"Start/Stop Debug Session" or click icon on the tool bar to exit the debugging status.

If the Senselock ELprogram exits the debugging status by executing the _exit() system function, there will show up a dialog box for execution result, which is the data returned from Senselock ELas shown in Figure 5-13.



Length: 0x0b Bytes State:0x00, Success

```
0000 68 65 6C 6C 6F 20 77 6F 72 6C 64
```

Figure 5-13 Execution Result Dialog Box

The execution result dialog box displays the output data contents set for the last time by the `_set_response()` system function. It has the same meaning as the data returned by calling the actual hardware.

## 5.2. Direction for Raisonance Rkit Usage

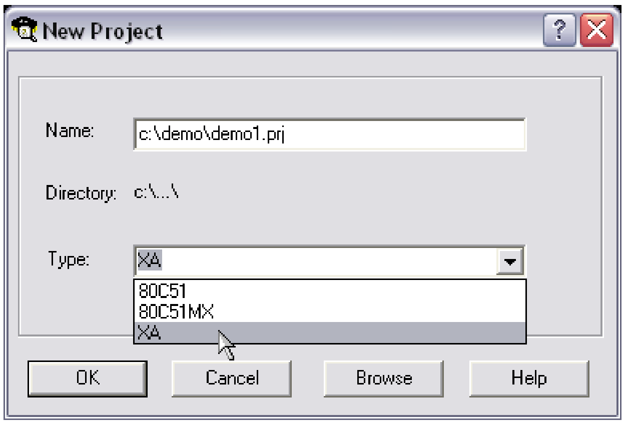
Rkit is a compiler software offered by Raisonance. Unlike Keil C51, the beta version Rkit offered by Raisonance cannot be used in SenseLock EL, so you need to purchase the official version of the software.

The objective codes compiled by Rkit can only be run in Senselock EL2.3. If you wish to be compatible with earlier Senselock ELhardware version, it is recommended to choose Keil C51 or Skit to compile your code.

### 5.2.1. Project Creation and Management

A typical Rkit project includes project file **.prj*, several C code files (including header files), Senselock ELsystem function header file *ses_v3.h* and Senselock ELsystem function library file *ses.lib*. It is more complex to create a Rkit project than in Keil C51.

1. Start the software, select the menu "Project"→"New", choose Type as XA and set the location for project saving and its name, as shown in Figure 5-14:



Name: c:\demo\demo1.prj

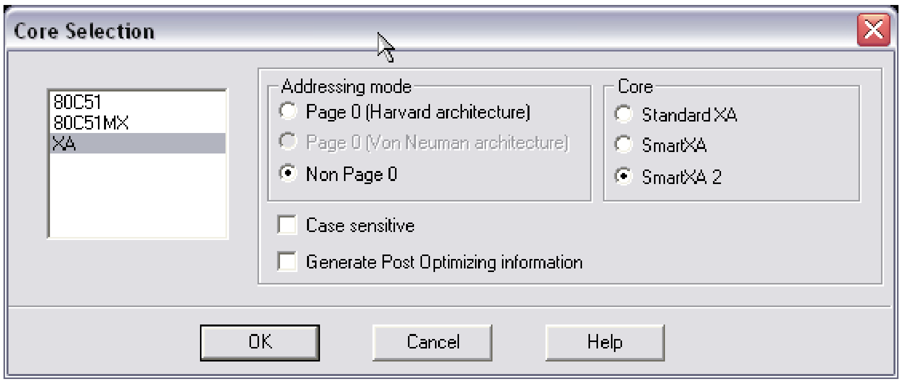
Directory: c:\...\

Type: XA

80C51
80C51MX
XA

Figure 5-14 Naming Project

2. In the dialog box of "Core Selection" that follows, choose "Addressing mode" as "Non Page 0" and "Core" as "SmartXA2" as shown in Figure 5-15:



80C51
80C51MX
XA

Addressing mode
- Page 0 (Harvard architecture)
- Page 0 (Von Neuman architecture)
- [✓ Non Page 0]

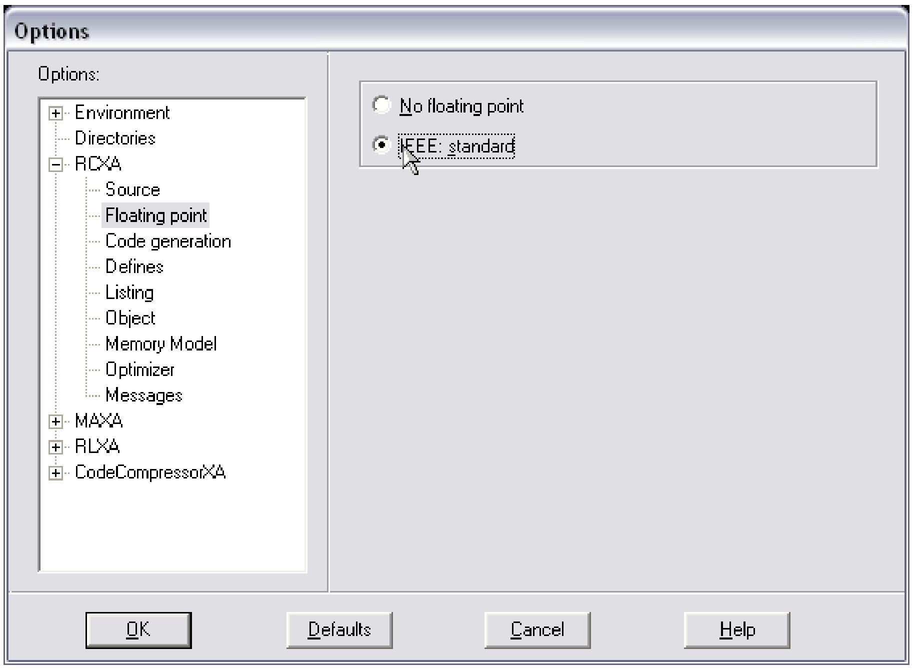
Core
- Standard XA
- SmartXA
- [✓ SmartXA 2]

☐ Case sensitive
☐ Generate Post Optimizing information

Figure 5-15 Core Selection

3. Configure project options. Choose the menu "Options"→"Project" and make the following configuration:

a) Spread the RCXA tree and configure the compilation options. Choose the "Floating Point" node and "IEEE: standard" option as shown in Figure 5-16:



No floating point [✓ IEEE: standard]

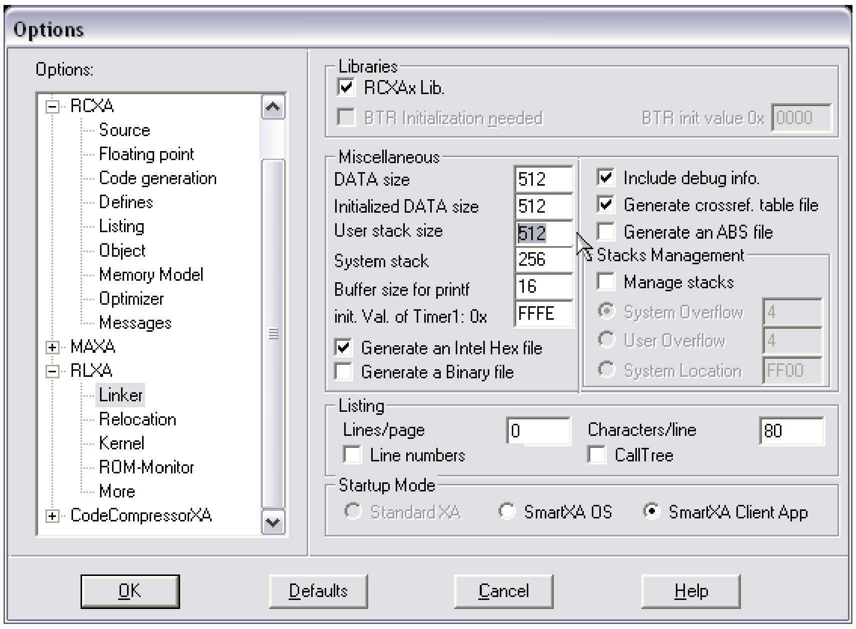
Figure 5-16 Compilation Option Configuration — Floating Point

b) Select the “Code generation” node and choose the option “Do not insert extra bytes in structures for alignment” to address the issue of byte alignment;

c) Select the “Memory Model” node and choose the options “Large” and “Functions in user mode”;

d) Spread the RLXA tree and configure Linker options. Select the “Linker” node, choose “SmartXA Client APP” from the “Startup Mode” field, and set “User stack size” as desired size. It is recommended to set it to 512 bytes[12] as shown in Figure 5-17. However, if VM and multi-XA programs need to share the memory, then set “Intialized DATA size” as 0.

[12] User Stack, namely user stack management, is rather complicated. The parameters and local variables of any function will seize user stack. On one hand, we suggest saving stack as much as possible, such as reducing the use of local variables, avoiding passing in parameters that occupy a large amount of memory in function call; on the other hand, the size of user stack shall be relatively sufficient, because the compiler cannot check whether the stack overflows. It shall be noted that the stack space divided in Rkit can not be used as user memory. In other words, if 512 bytes are allocated for user stack, the available memory of users will decrease by 512 bytes.


Options

Options:
- RCXA
  - Source
  - Floating point
  - Code generation
  - Defines
  - Listing
  - Object
  - Memory Model
  - Optimizer
  - Messages
- MAXA
- RLXA
  - Linker
  - Relocation
  - Kernel
  - ROM-Monitor
  - More
- CodeCompressorXA

Libraries
- [✓] RCXAx Lib.
- [ ] BTR Initialization needed — BTR init value 0x 0000

Miscellaneous

| | |
|---|---|
| DATA size | 512 |
| Initialized DATA size | 512 |
| User stack size | 512 |
| System stack | 256 |
| Buffer size for printf | 16 |
| init. Val. of Timer1: 0x | FFFE |

- [✓] Generate an Intel Hex file
- [ ] Generate a Binary file
- [✓] Include debug info.
- [✓] Generate crossref. table file
- [ ] Generate an ABS file

Stacks Management
- [ ] Manage stacks
- System Overflow 4
- User Overflow 4
- System Location FF00

Listing
- Lines/page 0
- Characters/line 80
- [ ] Line numbers
- [ ] CallTree

Startup Mode
Standard XA SmartXA OS [✓ SmartXA Client App]

Figure 5-17 Linker Option Configuration

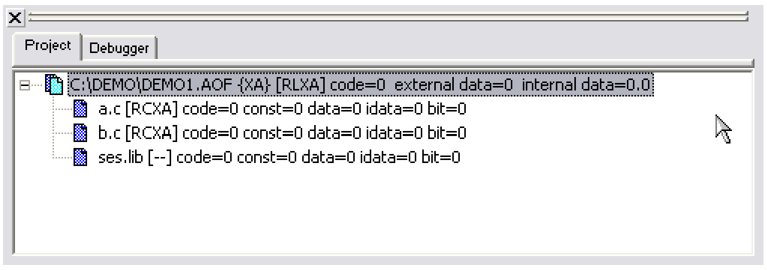
e) Choose OK to save project configuration;

4. Make sure *ses_v3.h* is already copied to the current project directory or to the directory of "*%RIDE%\ INC"*. Or else there will occur an error of failure to find the header file *ses_v3.h* in the middle of compilation. *Ses_v3.h* is located under the directory of "*%SDK%\IDE\RKIT\INC"*;

5. Add the C source codes written by yourself to the project. Choose the menu "Project"→"Add node Source/Application" to add the specified file to the project;

6. Add the SES library file *ses.lib* into the project. This file is located under the directory of "*%SDK%\IDE\RKIT\LIB"*;

It shall be noted that Rkit requires *ses.lib* to be located at the bottom most position of the project list; or else linker error will occur. Right click the file (such as *ses.lib*) to be moved and choose "Mode node"→"Up" or "Down" on the popup menu to move. The correct file list shall be similar to that in Figure 5-18.



```
C:\DEMO\DEMO1.AOF {XA} [RLXA] code=0  external data=0  internal data=0.0
    a.c [RCXA] code=0 const=0 data=0 idata=0 bit=0
    b.c [RCXA] code=0 const=0 data=0 idata=0 bit=0
    ses.lib [--] code=0 const=0 data=0 idata=0 bit=0
```

Figure 5-18 Project File List Demo

### 5.2.2. Code Compilation and Debugging

Choose the menu "Project"→"Build all" to compile and link the project. If compilation is successful, a HEX file will be generated accordingly in the project directory.

Up till now, we have not yet offered simulated debugging environment in the Rkit compiler.

To trace and debug the codes, it is recommended to debug codes in Skit or Keil before compiling them with Rkit in the end.

## 5.3. Senselock Skit

Skit will be described independently in another manual. For details, please contact Senselock.

## 5.4. Executable File Format

Whatever compiler is used, the compilation result is a HEX file in text format. To execute it in Senselock EL requires that it should first be converted into binary format, abbreviated as BIN file.

For the convenience of conversion, SDK offers tool software: *hexbin.exe*. The usage of this software is like this:

```
hexbin.exe [file.hex] [file.bin] i 1
```

The API and development testing tool of the current version can process HEX files directly. For example, when the `S4WriteFile()` function is used to write the EXF into SenseLock EL, you can input directly the HEX file and specify the file type as `S4_HEX_FILE` (VM executable file) or `S4_XA_HEX_FILE` (XA executable file). In this case, API will conduct automatically the conversion from HEX format to BIN format.

# 6. Direction for Tool Software Usage

To make it easy to test Senselock EL hardware and develop application programs, *%SDK%\Tools* directory offers some common tool software as shown in Table 6-1.

Table 6-1 Tool Software Introduction

| Tool software name | Purpose | File name | Hardware vision |
|---|---|---|---|
| Development test tool | Senselock ELapplication program test tool carries out developer level management and user level call of Senselock ELdevice. | *DevTest.exe* | *Standalone (above and v2.0)* *Network (above v2.0.5)* |
| User test tool | Conducts basic tests on hardware devices, designed to get device basic information and check whether the device is running normally. | *UserTest.exe* | *Standalone (above and v2.0)* *Network (above v2.0.5)* |
| Network version servise program | For Senselock ELnetwork version's service program, please refer to Chapter 7 "Direction for Network Version Usage". | *e4nsrv.exe* | *Network (above v2.0.5)* |
| Network version service management program | For Senselock ELnetwork version's service management program, please refer to Chapter 7 "Direction for Network Version Usage". | *e4nmgr.exe* | *Network (above v2.0.5)* |
| Network version test tool | Test tool designed for Senselock ELnetwork version's service program. | *NetUserTest.exe* | *Network (above v2.0.5)* |

The device being initialized by the current version development tools can be operated by previous development tools, and vice versa. If you are using EL v3.0, it is not allowed to initialize by tools in v2.3.2 development kit.

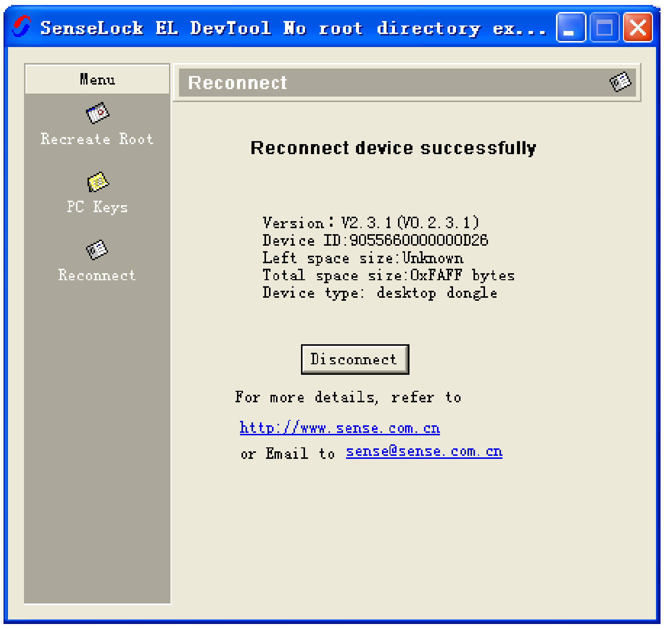
## 6.1. Development Test Tool

The development test tool is the most common tool software whose basic functions include:

- Device connection. Connect and disconnect the device and view device information.
- Directory management. Create directory, delete (root directory) and clear (sub-directory).
- PIN management. Modify and check developer PIN, user PIN.
- File management. Create and download files. For RSA key files, it can generate RSA public and private key pair using the hardware device directly
- Call test. Test to call the executable files downloaded to Senselock ELhardware.
- Cryptography algorithm. The software can generates RSA key pair and execute cryptography algorithm.

### 6.1.1. Device Reconnection

When the tool software is started, it will search automatically the Senselock ELdevices connected in the system. If no device is found, the software will give a warning. Click “Reconnect” to further display hardware information as shown in Figure 6-1:



Reconnect device successfully

```
Version: V2.3.1(V0.2.3.1)
Device ID:9055660000000D26
Left space size:Unknown
Total space size:0xFAFF bytes
Device type: desktop dongle
```

For more details, refer to

http://www.sense.com.cn

or Email to sense@sense.com.cn

Figure 6-1 Display Device Information

The tool software opens the dongle in “shared mode”, so in the middle of device connection, other software can still access SenseLock EL. Click “Disconnect” to close the connection of the tool software with the dongle.

### 6.1.2. Device Reset

The most fundamental directory management involving creating or deleting the root directory. In the software, choose “Recreate Root” to perform root-directory-related operations: create root directory, delete root directory, delete root directory and create new root directory (by default).

As for the device shown in Figure 6-1, no root directory is not yet created in the hardware. After reset, new root directory will be created.

When creating root directory, it is required to input the device ID as in Figure 6-2:

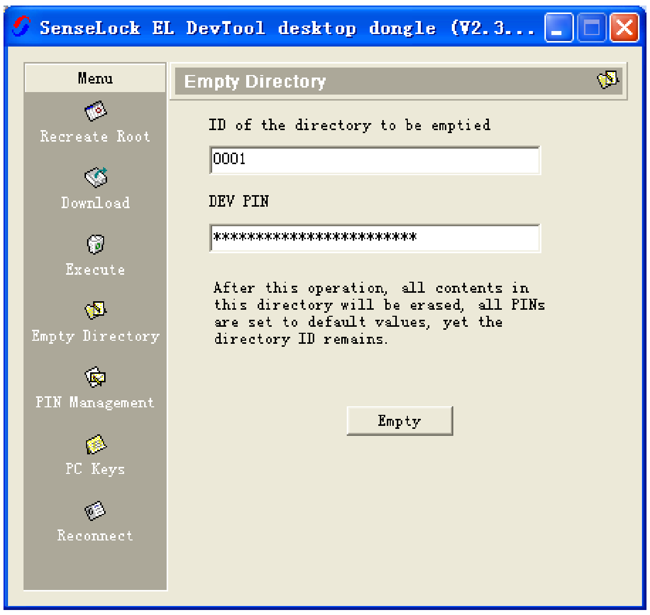
For network version, setting module anthorization mode is essential, Figure 6-3:

For the device where the root directory already exists, when you click the button “Recreate Root” to recreate it , it is necessary to input the developer level PIN for the original root directory; otherwise there is no way to achieve this. If you wish to delete the root directory directly, you need to choose the box “Do not recreate after erase root dir”.

After the directory is created (deleted, recreated or cleared), the developer level PIN and user PIN of the directory will be restored to its default value. For the detailed description of default PIN value and security, please refer to 1.3.1 “PIN” and 7.1.2 “Network Local PIN”.

Attention:
1. Only one level of sub-directory is supported in the development test tool. To use two levels of sub-directory, you need to write programs yourself.
2. For the network version, after the sub-directory is cleared, its authorization count will be restored to a

default value, namely, 10. For the detailed information of authorization count, please refer to Chapter 7 "Direction for Network Version Usage".



ID of the directory to be emptied

0001

DEV PIN

************************

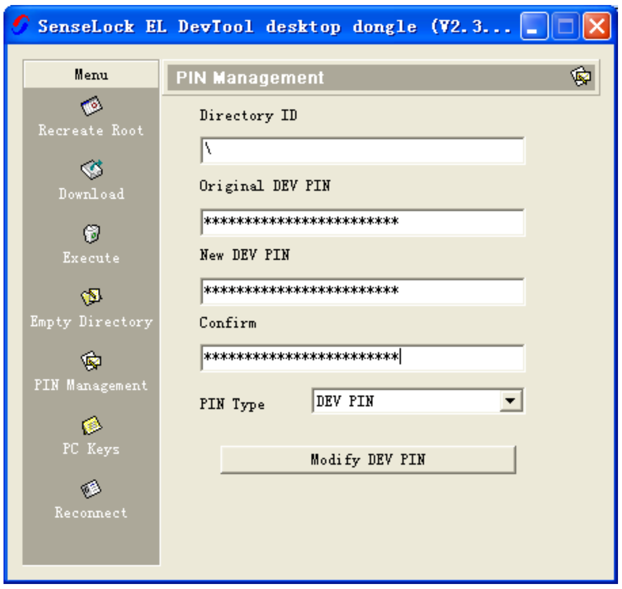
After this operation, all contents in this directory will be erased, all PINs are set to default values, yet the directory ID remains.

Figure 6-2 Clear Sub-directory Contents

### 6.1.3. PIN Management

PIN (also known as password) is the foundation for Senselock ELfile system security. For the detailed description of the PIN code, please refer to 1.3.1 "PIN" and 7.1.2 "Network Dongle PIN".

On the "PIN management" of the development test tool, you can modify the developer's PIN and user's PIN of Senselock EL. The interface for PIN modification is as shown in Figure 6-4.



Directory ID

\

Original DEV PIN

************************

New DEV PIN

************************

Confirm

************************

PIN Type DEV PIN

Figure 6-4 PIN Code Management

Input the directory whose PIN code needs to be modified, choose PIN modification type and input old PIN and new PIN according to the prompt on the interface before clicking the "Modify DEV PIN" or "Modify User PIN" (The button will display different texts according to the selected PIN type) button to modify the corresponding PIN.

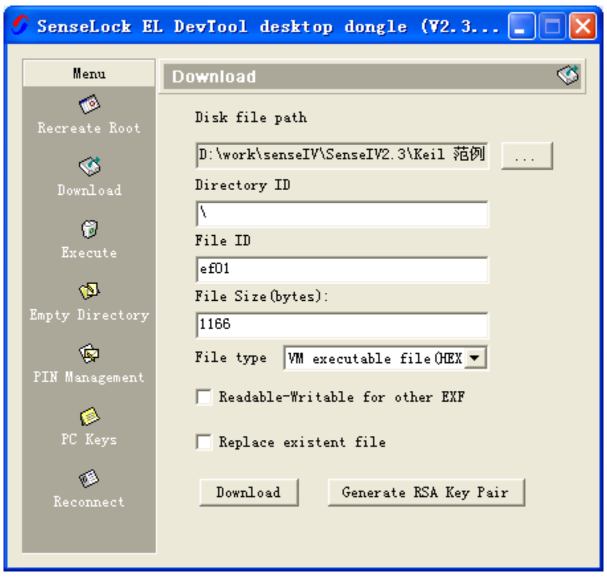
For network dongle, PIN of root directory is only available.

## 6.1.4. Download Files

The function of "Download" is to download disk files to the hardware. The current downloadable file types include executable files, data files and RSA key files. When the executable file is downloaded, it is required that the disk file should be in INTEL HEX format or binary format. The format required for the downloading of RSA files is `S4_RSA_PUBLIC_KEY` or `S4_RSA_PRIVATE_KEY`, which differs from the actual storage format in the hardware. The tool will conduct conversion automatically. For detailed information, please refer to Chapter 9 about the description of `S4WriteFile()` function.

When the downloading format is INTEL HEX, the downloading tool will convert the HEX file automatically into BIN file and save it in the current directory.

The "Download " page is shown in Figure 6-5:



Disk file path: D:\work\senseIV\SenseIV2.3\Keil 范例
Directory ID: \
File ID: ef01
File Size(bytes): 1166
File type: VM executable file(HEX
☐ Readable-Writable for other EXF
☐ Replace existent file

Figure 6-4 Download Files

When files are downloaded, it is necessary to choose the type of files to be downloaded and then click the "..." button to choose the corresponding disk files. After the disk files are selected, the contents of the "File size" input box will be automatically updated to the actual storage space required by the hardware. To create a new file, you can also input a larger value so that the file can be used repeatedly. After that, fill in the downloading directory and file name (the directory name and file name shall be a 2-byte hexadecimal number. For example, the file name in Figure 6-4 is EF01) and finally click the "Download" button to download the file to the hardware.

When the file type is set as executable file, you can choose the attribute of "Readable-Writable for other EXF". For the readable and writable attributes of the executable file, please refer to the description of the `S4WriteFile()` function.

When a file already exists, you can choose "Replace existent file" to update the file contents. The size of the file to be downloaded shall not exceed that of the file to be overwritten.

If the child directory does not exist when downloading files, a hint will remind you to build one.

For network dongle, downloading file is only available from root directory.

### 6.1.5. Clear Directory Content

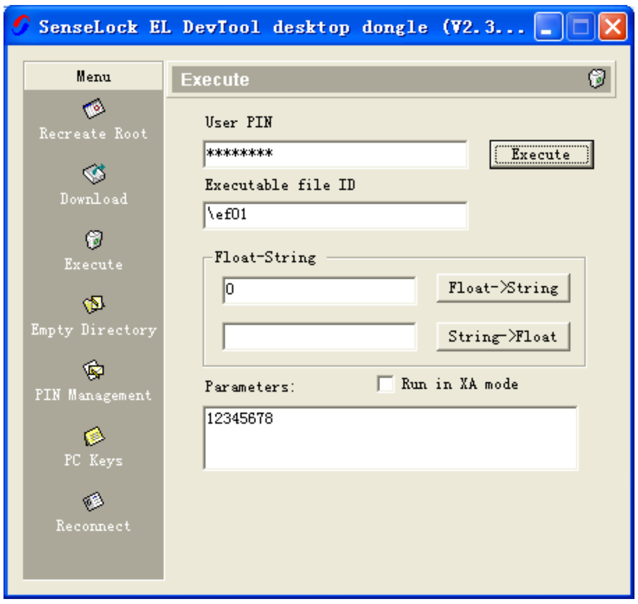
For network dongle, this feature is not applicable.

For standalone dongle, input the directory ID and developer PIN of directory to empty the directory and reinitiate default PIN as Figure 6-6.

### 6.1.6. Execute File

For network dongle, this feature is not applicable. Execution file in network dongle requires EL Network Test Tool.

The function of "Execute" is to execute a exe file stored in Senselock EL as shown in Figure 6-5;



User PIN: ********

Executable file ID: \ef01

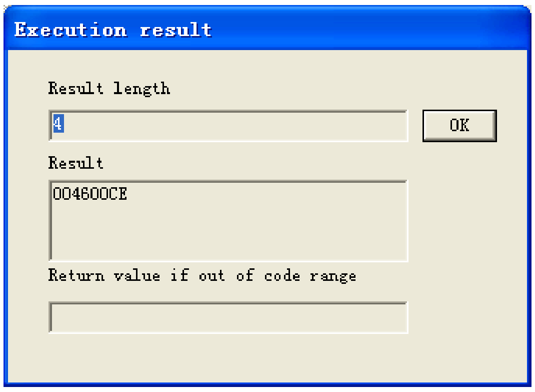
Float-String: 0

Parameters: 12345678

Figure 6-7 Executable File

When executing a file, User PIN is required. For the convenience of testing, "12345678" is preset value. And "Executable File ID" must be filled by full path name, for instance, "\000a". The "Parameter" must be hexadecimal value. Click

Click the "Execute" button and the tool will execute the specified executable file in the hardware and return execution result as shown in Figure 6-8:



Result length

4

Result

004600CE

Return value if out of code range

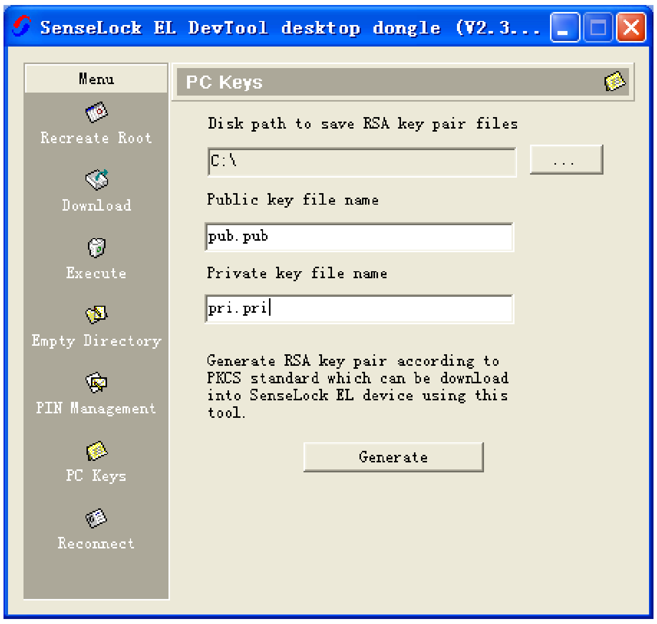
Figure 6-8 Execution Result

### 6.1.7. Autherization Management

For standalone dongle, this feature is not applicable.
In the tab, it is editable on network dongle overall autherrization, module autherisation information and mode, as shown in Figure 6-9.

### 6.1.8. PC Keys

The "PC keys" function is designed to generate RSA key pair files in `S4_RSA_PUBLIC_KEY` and `S4_RSA_PRIVATE_KEY` format on the local computer and download them to the hardware or for other purposes. Choose the directory where the files are to be saved, fill in the "public key file name" and "private key file name" and then click the "Generate" button to generate specified key pairs by software as shown in Figure 6-10;



Disk path to save RSA key pair files

C:\

Public key file name

pub.pub

Private key file name

pri.pri

Generate RSA key pair according to PKCS standard which can be download into SenseLock EL device using this tool.

Figure 6-10 PC Key

# 7. Direction for Network Version Usage

The network version can allow several software applications to access concurrently the same dongle connected to the server, namely, to share the dongle. By limiting the amount of software running concurrently at the client end, the purpose of controlling software use can be achieved. Unlike the desktop version, the network version needs not only to protect the software from illegal copying but also to limit strictly the amount of concurrently running software.

This chapter introduces Senselock ELnetwork version.

## 7.1. Introducing Senselock ELNetwork Version

### 7.1.1. Features

Compared with other network version dongles, the biggest benefit of Senselock ELnetwork version is that it guarantees, to the highest extent, authorization security on top of affording software protection with the same strength as desktop version SenseLock EL.

1. Software Protection Capability

   In terms of software protection technology, there is no difference between Senselock ELnetwork version and the desktop version. You can accomplish the development of software protection on the desktop version and apply it directly on the network version Senselock ELlater on. The only job that needs to be done is to re-compile your software.

   With SenseLock EL, you do not need to worry that the risk of piracy might increase when the software is switched from the desktop version to the network version; whereas this is a grave defect of other network version dongles.

2. Authorization Control Capability

   Senselock ELnetwork "ports" innovatively the authorization management module to the hardware. In the previous network dongles, the authorization number (the number of clients that can make concurrent access) is ultimately determined by the "network service program" running on the server. The modification of the network service program might render the authorization number control invalid. Senselock ELnetwork version is totally different: all the authorization control is done in the hardware. The network service program of it is responsible only for communication between the client and Senselock ELhardware, so there is no way to attack authorization number.

Network dongle has two types of authorization: Overall and Module Authorization. The program inside the dongle gains a module authorization before being executed. The module is the minimum authorization unit with an ID (0x0000-0x00ff) and a value (0x0000-0x00ff). Overall authorization stands for the total authorisable number regardless modules; the default value is 0; the max value is 255. It implys that no matter how many modules being set or what value being set, in process of using network dongle, the total authorized number must not exceed the Overall Authorization.

There are two authorization modes: 1 Process Mode. Obtaining a module's authorization means occupying an authorization of the module.   2. IP Mode. From the same IP address, obtaining any authorization of the same module means that sharing and occupying an authorization only.

### 7.1.2. Network Dongle PIN

Like the desktop version of SenseLock EL, the network version dongle also contains the developer level PIN, which is fully identical with that of Senselock ELdesktop version, maintaining each directory independently and offering local developer level access to the hardware device. For details, please refer to 1.3.2 “PIN”.

In terms of User PIN, there are some differences between the network version and the standalone version. For the network dongle, the root directory is available to use only, therefore, only root directory has Developer PIN and User PIN.

### 7.1.3. Network Dongle Module

The module of network dongle is defined logical unit by developers. Each one has to be set with authorized information.

Note:

In old versions, that a module is a child to first level directory.

A network dongle can contain 16 modules in maximum. Each module ID ranges from 0x0000-0xffff; each module’s authorization value ranges from 0x0000-0x00ff.

## 7.2. Using Network Version SenseLock EL

### 7.2.1. Installation and uninstallation of Network Dongle Service Program

The Network Dongle Service Program e4nsrv.exe is executable under Windows in the mode of service, managing ports, remote services and logging on daily basis for administration.

While installing SDK, e4nsrv.exe will be installed as well automatically. It is checkable by opening Computer Management to find a service named Elite4 Net Service.

If the serice does not appear in the service list, you could manually install by command e4nsrv –i. And the unstallation command is e4nsrv –u. Other options will be listed by typing e4nsrv -?.

Please refer to 7.2.2 to find out how to setup parameters.

### 7.2.2. Use of Network Dongle Service Management Tool

The service management tool, e4nmgr.exe is appled to set working parameters, to start/stop program, to display the working status, and to manage authorization upon client-side, as shown in Figure 7-1.

1. Parameter Setup

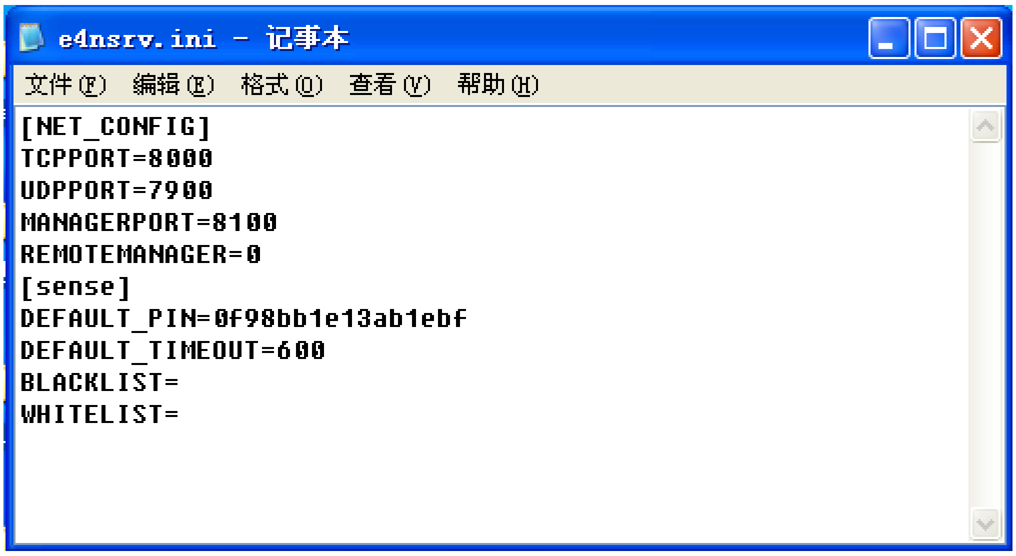
   The e4nmgr.exe is applied to set up the working paramenters on network dongle. Click button Sepup or menu Management -> Set Parameter, if the tool is running at the server of hosting services, as shown in Figure 7-2.

   If the tool is not running at the host server, it is only available to modify the item Management Tool Network Settings.

Figure 7-2 displays, the tool allows to setup Management Tool Network and Server Side Network with dongle itself.

In the Dongle Parameter Setting, if the device ID is input, the relavant parameters will be displayed including overdue, black-white list, as Figure 7-3.

Developers could modify the default User PIN by using the interface. After the completion of set-up, Click OK to save the info to the config file e4nsrv.ini under the directory preserving service as shown in Figure 7-4.



```
[NET_CONFIG]
TCPPORT=8000
UDPPORT=7900
MANAGERPORT=8100
REMOTEMANAGER=0
[sense]
DEFAULT_PIN=0f98bb1e13ab1ebf
DEFAULT_TIMEOUT=600
BLACKLIST=
WHITELIST=
```

2. Add Host

   After setting up the paramenter, it is ready to add the host. Click File-> Add Host, as the Service Management Tool working with Service on the same machine, IP must be set as 127.0.0.1, as Figure 7-5.

3. Start/Stop Services

   If the Service Management Tool is running on the same manchine with service, it is available to operate on service status by clicking Management->Start/Stop Services or hit shortcut botton on the tool bar, as in Figure 7-6. Otherwise, there is no way to make the same operation.

4. Connect Device

   After the service starts, it is operational on connecting device for obtaining the info of the dongle and its modules. Click File->Connect or directly hit the shortcut button on tool bar. Once the connection is successfully built, the tool will display the required info on dongles and its modules as Figure 7-7.

5. Apply / Release Authorisation (Kick Peer)

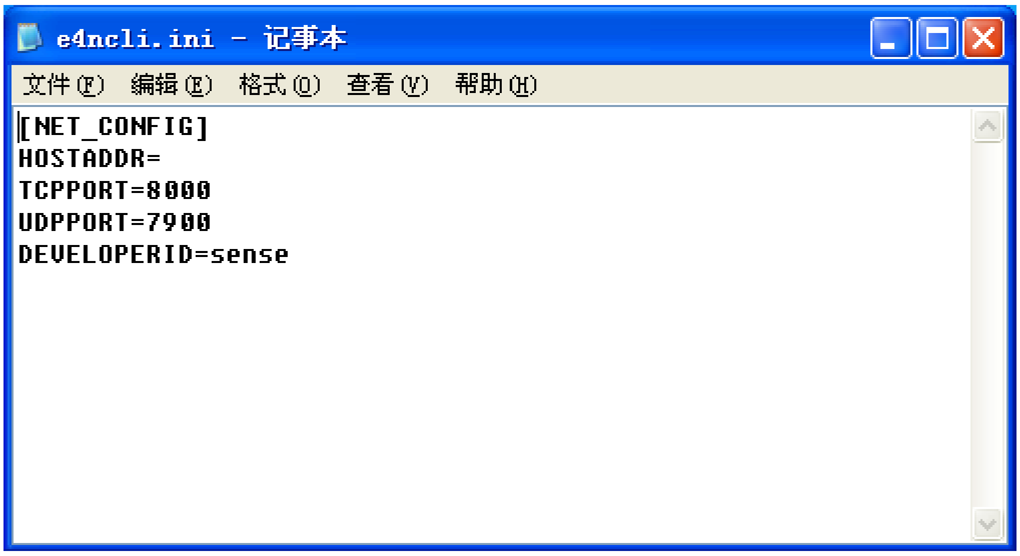
   As the servise starts and connects to device successfully (the info of dongle and its modules are displayed correctly), if any peer connects and applys for authorization, the tool will update automatically: display user's IP, applied modules, total authorization of the module, occupied authentication of the module, total authorization of device, occupied authorization of device, as Figure 7-8.

   If it is necceory to release the authorization by force on some chosen peer, Right Click->

Release Authorization, or hit the shortcut botton on tool bar as Figure 7-9.

### 7.2.3. Use of Network Dongle API

The network dongle API is used on accessing Elite EL network version remotely. Despite from S4Enum, S4Open and so on, other APIs were reserved to be compatible downwards, please refer to Chpater 9. Please set the client-side config e4ncli.ini correctly while using network dongle API, as Figure 7-10.



```
[NET_CONFIG]
HOSTADDR=
TCPPORT=8000
UDPPORT=7900
DEVELOPERID=sense
```

HOSTADDR item is IP address of server. If this item is empty, the client side will broadcase in mode of message to search for services by default; otherwise, the client side will do by assigned IP address.

DEVELOPERID is the itme of Developer ID, client-side must set this correctly to find the device.

The full sample is placed under the directory %SDK%/samples/api/net device.

# 8. SES Reference

SES(System Extension Service) is a group of functions supported by Senselock ELhardware. All the SES functions are declared in the header file:ses_v3.h. This header file also defines a group of macros together with some auxiliary functions to simplify the developing process. According to the different purpose, SES and macros can be divided to the following categories:

Attention: Some SES functions require specific hardware version. For more detail, please refer to the “Requirements” section of each SES’s reference.

Table 8-1 SES and MACRO

| Category | Comments |
|---|---|
| Flow Control | Please refer to 8.1 |
| Input/Output | Please refer to 8.2 |
| File Operation | Used to operate internal file of SenseLock EL, i.e. data file. For more detail, please refer to 8.3 |
| Mathematics | Support simple/double precision float calculation. Provide all the common mathematic functions defined in math.h of standard C language. For more detail, please refer to 8.4 |
| Cryptographic Algorithm | Provide SHA1、RSA、DES/TDES Cryptographic algorithms. For more detail, please refer to 8.5 |
| Memory Operation | Basic memory operation support. I.e.: dynamic memory allocation, memory copy etc. For more detail, please refer to 8.6 |
| Time | Timer and real time clock functions. For more detail, please refer to 8.7 |
| Macro and Auxiliary function | For more detail, please refer to 8.8 |
| Get Device Info | I.e. function used to get the GUID of the device. For more detail, please refer to 8.9 |

## 8.1. Flow Control

### 1). _exit

Exit Senselock ELEXF

```
void _exit(void)
```

**Parameters:**

None

**Return Values:**

None

**Remarks:**

Must use this function to terminate Senselock ELhardware program(EXF) and invoking this function anywhere will result in the termination of the EXF.

**Requirements:**

Hardware version :Senselock EL2.x

**Example Code:**

```
please refer to the sample code of _set_response()
```

## 8.2. Input/Output

### 2). _set_response

Set the data to be transmitted back to PC by Senselock ELEXF.

```
BYTE  _set_response(
  BYTE bLen,
  BYTE* pbBuff);
```

**Parameters:**

| | |
|---|---|
| *bLen* | [in] Length of returned data. |
| *pbBuff* | [in] Buffer address. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

The length of returned data cannot exceed that of the communication buffer.Namely, 250 bytes for desktop version and 244 for network version.

**Requirements:**

Hardware version :Senselock EL2.x

**Example Code:**

```
#include “ses_v3.h”

char buff[32] = “This is a test buffer.”;

void main()
{
  _set_response(sizeof(buff), (BYTE *)buff);
  _exit();
}
```

## 8.3. File Operation

This group of functions is used to operate the internal file system of Senselock ELdevice.

### 3). _open

Select and open the file specified.

```
BYTE _open(
  WORD wFid,
  HANDLE* pHandle);
```

**Parameters:**

| | |
|---|---|
| *wFid* | [in] File ID. |
| *pHandle* | [out] Handle to the file opened. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Senselock ELinternal program(EXF) can open up to 3 files concurrently. If you want to operate more files, unwanted file handles must be closed using _close().
You must check the Return value to see whether or not the file-open operation succeeds!

**Requirements:**

Hardware version :Senselock EL2.x

**Example Code:**

```
please refer to the sample code of _write().
```

### 4). _close

Close the file specified.

```
BYTE _close(HANDLE handle);
```

**Parameters:**

| | |
|---|---|
| *handle* | [in] Handle to the file to be closed |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
please refer to the sample code of _write().
```

## 5). _read

Read data from a opened file. Reading privilege must be satisfied according to the file security attributes. For more detail, please refer to the “Remarks” section of this table.

```
BYTE _read(
  HANDLE handle,
  WORD wOffset,
  BYTE bLength,
  BYTE* pbBuff);
```

**Parameters:**

| | |
|---|---|
| *handle* | [in] Handle to the file opened. |
| wOffset | [in] Reading offset |
| *bLength* | [in] Length of the data to be read,1~247bytes. |
| *pbBuff* | [out] Pointer to data buffer, used to store the data read. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

To read the content of the specified file, reading privilege MUST be satisfied according to the file security attributes:

- Data file, Public key file can be read by EXFs of the same directory freely.
- If the security attribute of an EXF has been set to “unreadable/unwrittable”,then it can not be read by other EXFs. Otherwise, it can be read by another EXF of the same type if they also reside in the same directory.(Same type means that a VM EXF can only be read by another VM EXF,and a XA EXF can only be read by another XA EXF).It’s highly recommended that you set the security attribute of all the EXFs to be “UNREADABLE/UNWRITTABLE”to enhance security unless any of them may be used for remote update purpose)
- Private key file can NEVER be read.

**Requirements:**

Hardware version :Senselock EL2.x

**Example Code:**

```
please refer to the sample code of _write().
```

## 6). _write

Write data to a file already opened, writing privilege must be satisfied according to the file security attributes. For more detail, please refer to the “Remarks” section of this table.

```
BYTE _write(
  HANDLE handle,
  WORD wOffset,
  BYTE bLength,
  BYTE* pbBuff);
```

**Parameters:**

| | |
|---|---|
| *handle* | [in] Handle to the file opened. |
| wOffset | [in] Writing offset |
| *bLength* | [in] Length of the data to be written,1~247bytes. |
| *pbBuff* | [in] Pointer to the data buffer, used to store data to be written in. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

To write data to the specified file, writing privilege MUST be satisfied according to the file security attributes:

- Data file, Public key file can be written by EXFs of the same directory freely.
- If the security attribute of an EXF has been set to “unreadable/unwrittable”, then it can not be written by other EXFs. Otherwise,it can be written by another EXF of the same type if they also reside in the same directory.(Same type means that a VM EXF can only be written by another VM EXF,and a XA EXF can only be written by another XA EXF).It’s highly recommended that you set the security attribute of all the EXFs to be “UNREADABLE/UNWRITTABLE”to enhance security unless any of them may be used for remote update purpose)
- Private key file can be written by EXFs of the same directory.

**Requirements:**

Hardware version :Senselock EL2.x

**Example Code:**

```
/* This demonstrates how to operate file in SenseLock EL. */
#include "ses_v3.h"

unsigned char buff[128];

void main()
{
  HANDLE hFile1;
  HANDLE hFile2;
  BYTE ret = 0;

  /* Open data file 0xa001. */
  ret = _open(0xa001, &hFile1);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
    _exit();
  }
```

```
    /* Read 128 bytes from file 0xa001. */
    ret = _read(hFile1, 0, 128,buff);
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
      _exit();
    }

    /* Open data file 0xa002. */
    ret = _open(0xa002, &hFile2);
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
      _exit();
    }

    /* Write what we read to file 0xa002. */
    ret = _write(hFile2, 0, 128, buff);
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
      _exit();
    }

    /* Close both files. */
    _close(hFile1);
    _close(hFile2);

    /* ret=0 indicates an successful operation.
      You can also send what we read to PC via
      _set_response(). */
    _set_response(1, &ret);

    /* Exit program. */
    _exit();

  }
```

## 7). _create

Create a new file and inherit the security attributes from it's parent file. For more detail, please refer to the "Remarks" section of this table.

```
BYTE _create(
  WORD wFid,
  WORD wSize,
  BYTE bFileType,
  BYTE bFlag,
  HANDLE * pHandle);
```

**Parameters:**

| | |
|---|---|
| *wFid* | [in] ID of the file to be created. |
| *wSize* | [in] Size of the file. |
| *bFileType* | [in] File type,possible values:<br>• SES_FILE_TYPE_EXE Executable（EXF）<br>• SES_FILE_TYPE_EXE_DATA Data file<br>• SES_FILE_TYPE_RSA_PUB RSA Public file<br>• SES_FILE_TYPE_RSA_SEC RSA Private file |
| *bFlag* | [in] Flag,possible values:<br>• CREATE_OPEN_ALWAYS<br>Open the file if it already exists, create a new file otherwise.<br>• CREATE_FILE_NEW<br>Create and open the file, return error if it already exists.<br>• CREATE_OPEN_EXISTING<br>Open an existing file. Similar to SES _open(). |
| *pHandle* | [out] Handle to the file if it's opened successfully. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Because this SES is designed specifically for remote operations, files created by it can only be accessed internally in SenseLock EL. In other words, one can't write a file created by this SES by using S4WriteFile() and PS4WriteFile() even if he has developer level PIN.

File created by this SES will inherit security attributes of its parent file(SES caller).So if one want to create a usable new file, the parent file(SES caller)must have its security attribute set to be "READABLE/WRITTABLE". Otherwise, a useless "dead file" will be created.(Senselock has admitted that's a design flaw, and it will be modified in the coming new version).

Executable(EXF)created by this SES must be enabled using SES _enable_exe() before it can be executed. EXF created by VM EXF is a VM EXF, EXF created by XA EXF is a XA EXF.

Caution: There is no data imported by executing this function. _write() is required to write data in the assigned file.

**Requirements:**

Hardware version:Senselock EL2.3

**Example Code:**

```
please refer to the sample code of _enable_exe().
```

## 8). _enable_exe

Enable EXFs created by SES _create().

```
BYTE _enable_exe(WORD wFid);
```

**Parameters:**

| | |
|---|---|
| *wFid* | [in] ID of the EXF to be enabled. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Effective only to EXFs created by SES _create().

**Requirements:**

Hardware version:Senselock EL2.3

**Example Code:**

```
/* This demonstrates how to create a new exe file and enable
it in SenseLock EL. */
#include "ses_v3.h"

void main()
{
  BYTE ret = 0;
  HANDLE hFile;

  /* Create a new exe file with file ID 0xa003. */
  ret = _create(0xa003, 1024, SES_FILE_TYPE_EXE,
          CREATE_FILE_NEW, &hFile);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
    _exit();
  }

  /* SHOULD set file content here….*/

  /* Enable the newly created file. */
  ret = _enable_exe(0xa003);

  _set_response(1, &ret);
  _exit();
}
```

## 8.4. Mathematics

This category includes all the commonly used mathematic functions and is the biggest group among SES.

If you want only single precision calculation result, SES functions are not absolutely necessary. You can just #include math.h and use functions defined in it. But, to VM EXF, using single precision float SES functions can boost calculation speed greatly. For XA EXF, those functions are provided just for compatibility purpose and will not effect calculation speed.

By contrary, double precision float calculation MUST use SES to accomplish due to the fact that the compiler can't support effective double type (keyword "double" maybe handled as "float", depending on the compiler setting). Double precision float in Senselock ELis represented using a struct(defined as DOUBLE_T) and thus you can't assign a value to it simply by using "=". There are three common ways to assign a double precision float: first one is by means of memory copy, i.e. get the byte array representation of the source in memory before assignment and then copy this array to destination DOUBLE_T; second, using SES to convert different format among single precision, integer, and double precision; third, using the auxiliary function which can convert a string to a double precision float.

About struct `DOUBLE_T`,the following problems should be paid attention to:

- ♦ This struct takes 8 bytes totally, and it has the same format as the double precision float real*8 stipulated in IEEE.
- ♦ `DOUBLE_T` uses Little-Endian,which is compiler-independent.

Single and double precision float SES functions have similar naming scheme, all the single precision functions have the suffix of 'f'.

### 9). _addf

Single precision float adding.

```
float _addf(
  float x,
  float y);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] augends. |
| *Y* | [in] addend. |

**Return Values:**

Sum of two single precision float.

**Remarks:**

None

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
After running,the result of sum will be 4.9 .
float sum, a = 3.2, b = 1.7;
sum = _addf(a, b);
```

## 10). _add

Double precision float adding.

```
BYTE _add(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  DOUBLE_T *py);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the augends. |
| *Py* | [in] Pointer to the addend. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

**Example Code:**

```
DOUBLE_T sum, a, b;
BYTE ret = 0;
/* assign value to a and b here… */
ret = _add(&sum, &a, &b);
```

## 11). _subf

Single precision float subtraction.

```
Float _subf(
  float x,
  float y);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] minuend. |
| *Y* | [in] subtrahend. |

**Return Values:**

Subtraction of two single precision floats.

**Remarks:**

None

**Requirements:**

Hardware version:Senselock EL2.x

### 12). _sub

Double precision float subtraction.

```
BYTE _sub(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  DOUBLE_T *py);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the minuend. |
| *Py* | [in] Pointer to the subtrahend. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

### 13). _multf

Single precision float multiplication.

```
float _subf(
  float x,
  float y);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Multiplicand. |
| *Y* | [in] Multiplier. |

**Return Values:**

Product of two single precision float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 14). _mult

Double precision float multiplication.

```
BYTE _mult(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  DOUBLE_T *py);
```

**Parameters:**

| | |
|---|---|
| *Presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the multiplicand. |
| *Py* | [in] Pointer to the multiplier. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 15). _divf

Single precision float division .

```
float _divf(
  float x,
  float y);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] dividend. |
| *Y* | [in] divisor. |

**Return Values:**

Quotient of two single precision float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 16). _div

Double precision float division.

```
BYTE _div(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  DOUBLE_T *py);
```

**Parameters:**

| | |
|---|---|
| *Presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the dividend. |
| *Py* | [in] Pointer to the divisor. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 17). _sinf

Sine of a single precision float.

```
float _sinf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Radian.. |

**Return Values:**

Sin of the radian.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 18). _sin

Sine of a double precision float.

```
BYTE _sin(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *Presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the radian. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 19). _cosf

Cosine of a single precision float.

```
float _cosf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Radian. |

**Return Values:**

Cosine of the radian.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 20). _cos

Cosine of a double precision float.

```
BYTE _cos(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the radian. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

### 21). _tanf

Tangent of a single precision float.

```
float _tanf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Radian. |

**Return Values:**

Tangent of the radian.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 22). _tan

Tangent of a double precision float.

```
BYTE _tan(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the radian. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 23). _asinf

Asin of a single precision float.

```
float _asinf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Sine value. |

**Return Values:**

Asin of the float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 24). _asin

Asin of a double precision float.

```
BYTE _asin(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the sine value. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 25). _acosf

Acos of a single precision float.

```
float _acosf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Cosine value. |

**Return Values:**

Acos of the float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 26). _acos

Acos of a double precision float.

```
BYTE _acos(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the cosine value. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 27). _atanf

Atangent of a single precision float.

```
float _atanf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Tangent value. |

**Return Values:**

Atangent of the float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 28). _atan

Atangent of a double precision float.

```
BYTE _atan(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the tangent value. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 29). _atan2f

Atangent of two single precision float’s quotient.

```
float _atan2f(
  float x,
  float y);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] dividend. |
| *Y* | [in] divisor. |

**Return Values:**

Atangent of two float’s quotient.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 30). _atan2

Atangent of two double precision float.

```
BYTE _atan2(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  DOUTLE_T *py);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the dividend. |
| *Py* | [in] Pointer to the divisor. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 31). _sinhf

Sinh of a single precision float.

```
float _sinhf(float x);
```

**Parameters:**

|   |   |
|---|---|
| *X* | [in] Radian. |

**Return Values:**

Sinh of the float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 32). _sinh

Sinh of a double precision float.

```
BYTE _sinh(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the radian. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 33). _coshf

Cosh of a single precision float.

```
float _coshf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Radian. |

**Return Values:**

Cosh of the float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 34). _cosh

Cosh of a double precision float.

```
BYTE _cosh(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the radian. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

### 35). _tanhf

Tanh of a single precision float.

```
float _tanhf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Radian. |

**Return Values:**

Tanh of the float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 36). _tanh

Tanh of a double precision float.

```
BYTE _tanh(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the radian. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 37). _ceilf

Calculates the ceiling of a single precision float.

```
float _ceilf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Floating-point value. |

**Return Values:**

This function returns a float value representing the smallest integer that is greater than or equal to x.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 38). _ceil

Calculates the ceiling of a single precision float.

```
BYTE _ceil(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the float parameter |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 39). _floorf

Calculates the floor of a single precision float.

```
float _floorf(float x);
```

**Parameters:**

| | |
|---|---|
| *x* | [in] Float parameter. |

**Return Values:**

This function returns a floating-point value representing the largest integer that is less than or equal to x.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 40). _floor

Calculates the floor of a double precision float.

```
BYTE _floor(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *Px* | [in] Pointer to the float parameter |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 41). _absf

Calculate the absolute value of a single precision float.

```
float _absf(float x);
```

**Parameters:**

|  |  |
|---|---|
| *X* | [in] Float parameter. |

**Return Values:**

Absolute value of x.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 42). _abs

Calculate the absolute value of a double precision float.

```
BYTE _abs(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the float parameter |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 43). _fmodf

Calculates the single precision floating-point remainder.

```
float _fmodf(
  float x,
  float y);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Dividend. |
| *Y* | [in] Divisor. |

**Return Values:**

The floating-point remainder of x / y.

**Remarks:**

The _fmodf function calculates the floating-point remainder f of x / y such that x = i * y + f, where i is an integer, f has the same sign as x, and the absolute value of f is less than the absolute value of y.

**Requirements:**

Hardware version:Senselock EL2.x

## 44). _fmod

Calculates the double precision floating-point remainder.

```
BYTE _fmod(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  DOUBLE_T *py);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the dividend. |
| *py* | [in] Pointer to the divisor. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

The fmod function calculates the floating-point remainder f of x / y such that x = i * y + f, where i is an integer, f has the same sign as x, and the absolute value of f is less than the absolute value of y.

**Requirements:**

Hardware version: Senselock EL2.3

## 45). _expf

Calculates the exponential.

```
float _expf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Exponent. |

**Return Values:**

This function returns the exponential value of the floating-point parameter, x, if successful.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 46). _exp

Calculates the exponential .

```
BYTE _exp(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the exponent |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 47). _logf

Calculates logarithms of a single precision float.

```
float _logf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Float parameter. |

**Return Values:**

Logarithm of the single precision float x.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 48). _log

Calculates logarithms of a double precision float.

```
BYTE _log(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the float parameter |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None..

**Requirements:**

Hardware version: Senselock EL2.3

## 49). _log10f

Returns the base 10 logarithm of a specified single precision float.

```
float _log10f(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Float parameter. |

**Return Values:**

The base 10 logarithm of a specified single precision float.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 50). _log10

Returns the base 10 logarithm of a specified double precision float.

```
BYTE _log10(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the float parameter |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None..

**Requirements:**

Hardware version: Senselock EL2.3

## 51). _sqrtf

Calculates the square root of a single precision float.

```
float _sqrtf(float x);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Float parameter. |

**Return Values:**

The square root of the single precision float, x.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 52). _sqrt

Calculates the square root of a double precision float.

```
BYTE _sqrt(
  DOUBLE_T *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the float parameter |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None..

**Requirements:**

Hardware version: Senselock EL2.3

## 53). _powf

Calculates x raised to the power of y where both x and y are of single precision float.

```
float _powf(
  float  x,
  float  y);
```

**Parameters:**

| | |
|---|---|
| *X* | [in] Base. |
| *Y* | [in] Exponent. |

**Return Values:**

The value of $x^y$.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 54). _pow

Calculates x raised to the power of y where both x and y are of double precision float.

```
BYTE _pow(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  DOUBLE_T *py);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the base. |
| *py* | [in] Pointer to the exponent |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None..

**Requirements:**

Hardware version: Senselock EL2.3

Attention: the following functions have only double float version.

### 55). _modf

Divide a double precision float to integral part and fractional part.

```
BYTE _modf(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  DOUBLE_T *intptr);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the fractional part. |
| *px* | [in] Pointer to the double to be divided. |
| *intptr* | [out] Pointer to the integral part. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None..

**Requirements:**

Hardware version: Senselock EL2.3

## 56). _frexp

Divide a double precision float to the product of a fraction and power of 2., please refer to the “Remarks” section of this table.

```
BYTE _frexp(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  int *expptr);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the fraction part. |
| *px* | [in] Pointer to the double to be divided. |
| *expptr* | [out] Pointer to the exponent |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This function calculate presult, expptr such that:

$*px = *presult \times 2^{(*expptr)}$

, where $1 > *presult >= 0.5$

**Requirements:**

Hardware version: Senselock EL2.3

## 57). _ldexp

Calculate the power of a double precision float and 2, please refer to “Remarks” section of this table.

```
BYTE _ldexp(
  DOUBLE_T *presult,
  DOUBLE_T *px,
  int exp);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the float parameter |
| *exp* | [in] Exponent of 2. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

The calculation formula is like that:

$*presult = *px \times 2^{(exp)}$

**Requirements:**

Hardware version: Senselock EL2.3

## 58). _fdcmp

Compare two double precision floats.

```
char _fdcmp(
  DOUBLE_T *px,
  DOUBLE_T *py);
```

**Parameters:**

| | |
|---|---|
| *px* | [in] Pointer to the first float parameter. |
| *py* | [in] Pointer to the second float parameter. |

**Return Values:**

Return comparison result, three possible values:

- `1  means:*px > *py`
- `0  means:*px = *py`
- `-1 means:*px < *py`

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 59). _dtof

Type conversion: from double precision to single precision float.

```
BYTE _dtof(
  float    *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the single precision float.. |
| *px* | [in] Pointer to the double precision float.. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 60). _ftod

Type conversion: from single precision to double precision float.

```
BYTE _ftod(
  DOUBLE_T *presult,
  float    *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the double precision float.. |
| *px* | [in] Pointer to the single precision float.. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 61). _dtol

Type conversion: from double precision float to 32 bits signed integer （long）.

```
BYTE _dtol(
  long   *presult,
  DOUBLE_T *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the double precision float.. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 62). _altod

Type conversion: from 32 bits signed integer(long)to double precision float.

```
BYTE _altod(
  DOUBLE_T  *presult,
  long   *px);
```

**Parameters:**

|  |  |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the 32 bits signed integer. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 63). _lltod

Type conversion: from 32 bits unsigned integer（long）to double precision float.

```
BYTE _lltod(
  DOUBLE_T  *presult,
  DWORD  *px);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the result. |
| *px* | [in] Pointer to the 32 bits unsigned integer. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version: Senselock EL2.3

## 8.5. Cryptographic Algorithm

Senselock ELprovides four categories of fundamental cryptographic algorithms: Asymmetric cipher algorithm(RSA), Symmetric cipher algorithm(DES/TDES), hash algorithm(SHA1) and real random number generator. These cryptographic algorithms can not only be used in software copyrights protection, but more importantly, can help developers to expand their softwares to network.. For example, they can be used together to implement a capable Remote Update platform.

To use these cryptographic algorithm functions properly, one need to have some basic understanding about cryptographic algorithm and application. This manual is not professional material for cryptography study and only describes the algorithms and protocols involved by our device briefly. To know more deeper knowledge, please refer to the relevant books in this field.

All the cryptographic algorithm has corresponding software version implementation. For example, you can use software-version functions to encrypt data, and then use corresponding SES of internal hardware to decrypt them. For more detail about software version cryptographic algorithms, please refer to the “Appendix A: Cryptographic Algorithms Reference”.

One special point: some of the cryptographic algorithm SES functions are kept only for compatibility purpose. Those functions have worse common-usability compare to their new version but have no security problem. To get better technology support in the future, it’s NOT recommended to use them in your program. All of those functions will be commented explicitely in this document.

## 64). _tdes_enc

TDES Encryption, ECB mode.

```
BYTE _tdes_enc(
  BYTE *pbKey,
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *pbKey* | [in] 16 octets DES key. |
| *bLen* | [in] Length of the data to be encrypted 1~248. |
| *pbData* | [in,out] Plaintext to be encrypted when input, ciphertext when output. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This function accomplish triple-DES encryption in ECB mode. It will divide the plaintext by eight octets a block, and generate a ciphertext block of the same length for every block independently. There is no relation between different blocks. If one want to use different encryption mode i.e. CBC, please encapsulate other code based on this function yourself or please refer to the sample code of Senselock ELSDK.

Please try to make the length of input plaintext to be multiple of eight, or try to do some padding to meet the requirement. Otherwise, this function will do some padding to plaintext automatically, and this will make the decrypted result contains some padding data that may lead to confusion.

The length of ciphertext is always multiple of 8, and it maybe longer than that of the plaintext(if the latter is not multiple of 8). Because the plaintext and the ciphertext use the same buffer, the former will be overwritten by the latter. So if you want to use the plaintext after calling this SES, please backup it to other memory block. In the other hand, you need sufficient buffer to hold the output ciphertext to avoid possible memory leak since the ciphertext maybe longer than the plaintext.

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
please refer to the sample code of _tdes_dec().
```

## 65). _tdes_dec

TDES decryption in ECB mode, paired with `_tdes_enc()`.

```
BYTE _tdes_dec(
  BYTE *pbKey,
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *pbKey* | [in] 16 octets DES key. |
| *bLen* | [in] Length of the data to be decrypted, 8~248,must be multiple of eight, refer to the "Remarks" section. |
| *pbData* | [in,out] Ciphertext when input, plaintext when output. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This function accomplish triple-DES decryption in ECB mode. It will divide the ciphertext by eight octets a block, and generate a plaintext block of the same length for every block independently. There is no relation between different blocks. If one want to use different decryption mode i.e. CBC, please encapsulate other code based on this function yourself or please refer to the sample code of Senselock ELSDK.

The length of ciphertext genereated by triple-DES encryption is always the multiple of eight, so this function demands inputting ciphertext with the length to be multiple of eight.

Because the ciphertext and the plaintext use the same buffer, the former will be overwritten by the latter. So if you want to use the ciphertext after calling this SES, please backup it to other memory block..

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
/*  This demonstrate how to use Triple-DES algorithm to
  encrypt some text and decrypt it later.
*/

#include "ses_v3.h"
#include "string.h"

char plaintext[24] = "This is some test data.";
BYTE ciphertext[24] = "";
BYTE buff[24] = "";
BYTE deskey[16] = {0x23,0x5a,0xb8,0x91,0xfc,0xe2,0x6c,0x9a,
         0x3a,0x98,0x34,0x8e,0x1d,0xaa,0x97,0xe1};

void main()
{
  BYTE ret = 0;

  /* Encrypt…*/
  memcpy(ciphertext, plaintext, 24);
```

```
    ret = _tdes_enc(deskey, 24, ciphertext);
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
    }

    /* Decrypt… */
    memcpy(buff, ciphertext, 24);
    ret = _tdes_dec(deskey, 24, buff);
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
    }
    /* Now the content of buff should be the same as
    plaintext. */

    _set_response(1,&ret);
    _exit();

  }
```

## 66). _des_enc

DES encryption in ECB mode.

```
BYTE _des_enc(
  BYTE *pbKey,
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *pbKey* | [in] 8 octets DES key. |
| *bLen* | [in] Length of the data to be encrypted, 1~248. |
| *pbData* | [in,out] Plaintext to be encrypted when input, ciphertext when output. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This function accomplish DES encryption in ECB mode. It will divide the plaintext by eight octets a block, and generate a ciphertext block of the same length for every block independently. There is no relation between different blocks. If one want to use different encryption mode i.e. CBC, please encapsulate other code based on this function yourself or please refer to the sample code of Senselock ELSDK.

This function uses key of eight octets, effective bits of which is 56 bits and is considered to be not secure enough. Please try to use `_tdes_enc()` which has 112 bits effective key instead. `_tdes_enc()` is secure enough and since the calculation is done by the DES/TDES accelerator in Senselock ELhardware, there will be no difference in performance/speed between DES and TDES.

Please try to make the length of input plaintext to be multiple of eight, or try to do some padding to meet the requirement. Otherwise, this function will do some padding to plaintext automatically, and this will make the decrypted result contains some padding data that may lead to confusion.

The length of ciphertext is always multiple of 8, and it maybe longer than that of the plaintext(if the latter is not multiple of 8). Because the plaintext and the ciphertext use the same buffer, the former will be overwritten by the latter. So if you want to use the plaintext after calling this SES, please backup it to other memory block. In the other hand, you need sufficient buffer to hold the output ciphertext to avoid possible memory leak since the ciphertext maybe longer than the plaintext.

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
please refer to the sample code of_des_dec().
```

## 67). _des_dec

DES decryption in ECB mode. Paired with `_des_enc()`.

```
BYTE _des_dec(
  BYTE *pbKey,
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *pbKey* | [in] DES key of 8 octets. |
| *bLen* | [in] Length of the data to be decrypted, 8~248, must be multiple of 8, please refer to “Remarks” section. |
| *pbData* | [in,out] Ciphertext when input, plaintext when output. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This function accomplish triple-DES decryption in ECB mode. It will divide the ciphertext by eight octets a block, and generate a plaintext block of the same length for every block independently. There is no relation between different blocks. If one want to use different decryption mode i.e. CBC, please encapsulate other code based on this function yourself or please refer to the sample code of Senselock ELSDK.

This function uses key of eight octets, effective bits of which is 56 bits and is considered to be not secure enough. Please use `_tdes_enc()` which has 112 bits effective key instead. `_tdes_enc()` is secure enough and since the calculation is done by the DES/TDES accelerator in Senselock ELhardware, there will be no difference in performance/speed between DES and TDES.

The length of ciphertext genereated by DES encryption is always the multiple of eight, so this function demands inputting ciphertext with the length to be multiple of eight.

Because the ciphertext and the plaintext use the same buffer, the former will be overwritten by the latter. So if you want to use the ciphertext after calling this SES, please backup it to other memory block..

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
/* This demonstrate how to use DES algorithm to encrypt
  some text and decrypt it later.
   This sample is almost the same as that using DES.
*/

#include “ses_v3.h”
#include “string.h”

char plaintext[24] = “This is some test data.”
BYTE ciphertext[24] = “”;
BYTE buff[24] = “”;
BYTE deskey[8] = {0x23,0x5a,0xb8,0x91,0xfc,0xe2,0x6c,0x9a};
```

```
void main()
{
  BYTE ret = 0;

  /* Encrypt…*/
  memcpy(ciphertext, plaintext, 24);
  ret = _des_enc(deskey, 24, ciphertext);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
  }

  /* Decrypt… */
  memcpy(buff, ciphertext, 24);
  ret = _des_dec(deskey, 24, buff);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
  }
  /* Now the content of buff should be the same as
  plaintext. */

  _set_response(1,&ret);
  _exit();

}
```

## 68). _sha1_init

SHA1initialization. SHA1 is completed using a group of functions. For more detail , please refer to “Remarks” section.

```
BYTE _sha1_init(HASH_CONTEXT *pContext);
```

**Parameters:**

| | |
|---|---|
| *pContext* | [in] HASH context. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This function initialize HASH context.
A typical HASH procedure includes:

- Initalize context using `_sha1_init()`;
- Add data, this can be achieved by calling `_sha1_update()` several times;
- Get hash result using `sha1_ final()`,purge the hash context at the same time;

Function group `_sha1_xxx()` need to maintain a buffer in the system, which will be used for the completion of RSA signagture verification. For detail, please refer to the sample code of `_rsa_veri()`.
Current version of Senselock ELdoesn’t allow multi HASH processes to run concurrently. Namely, one HASH process can run only after the previous one has completed. For example, the following procedure will not get corret results:

```
…
HASH_CONTEXT hctx1, hctx2;
BYTE message1[] = “test1”;
BYTE message2[] = “test2”;
BYTE len1 = 5;
BYTE len2 = 5;
BYTE digest1[20], digest2[20];

_sha1_init(&hctx1);
_sha1_init(&hctx2);

_sha1_update(&hctx1, message1, len1);
_sha1_update(&hctx2, message2, len2);

_sha1_final(&hctx1, digest1);
_sha1_final(&hctx2, digest2);
  …
```

Correct approach is:

```
…
HASH_CONTEXT hctx1, hctx2;
BYTE message1 = “test1”;
BYTE message2 = “test2”;
BYTE len1 = 5;
BYTE len2 = 5;
BYTE digest1[20], digest2[20];

_sha1_init(&hctx1);
_sha1_update(&hctx1, message1, len1);
_sha1_final(&hctx1, digest1);
```

```
_sha1_init(&hctx2);
_sha1_update(&hctx2, message2, len2);
_sha1_final(&hctx2, digest2);
  …
```

**Requirements:**

Hardware version: Senselock EL2.x

**Example Code:**

```
please refer to the sample code of _sha1_final().
```

## 69). _sha1_update

Adding SHA1 data can be achieved using a calling sequence. For detail, please refer to the "Remarks" section of `_sha1_init()`.

```
BYTE _sha1_update(
  HASH_CONTEXT *pContext,
  BYTE      *pbData,
  BYTE      bLen);
```

**Parameters:**

| | |
|---|---|
| *pContext* | [in] HASH Context. |
| *pbData* | [in] Data to be added this time. |
| *bLen* | [in] Length of the data to be added this time. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Use this function to add data to be hashed. A big data block can be divided to several smaller ones and added to hash function by calling `_sha1_update()` continuously.
Software version SHA1 has no limitation for the inputted data, SenseLock EL, in the other hand, can't exeed 8191 bytes for the totally inputted data due to its limitation of resource.

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
please refer to the sample code of _sha1_final().
```

## 70). _sha1_final

Get the result of SHA1 HASH and then purge the context. SHA1 is achieved by a group of functions. For more detail, please refer to the “Remark” section of `_sha1_init()`.

```
BYTE _sha1_final(
  HASH_CONTEXT *pContext,
  BYTE     *pbResult);
```

**Parameters:**

| | |
|---|---|
| *pContext* | [in] HASH Context. |
| *pbResult* | [out] 20 bytes hash result returned. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

After calling this SES, the system will also copy the result to a specific HASH buffer so as to be used in signature verification process. For relations between HASH buffer and signature verification, please refer to the “Remarks” section of`_rsa_veri()`.

pbResult must have sufficient storage so as to contain the HASH result.

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
/* This demonstrates how to get digest of a message. */

#include “ses_v3.h”

HASH_CONTEXT hctx;
BYTE message1[] = “This is the first part of test message.”;
BYTE message2[] = “This is the secont part of test message.”;
BYTE digest[20];

void main()
{
  BYTE ret = 0;

  ret = _sha1_init(&hctx);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
    _exit();
  }

  ret = _sha1_update(&hctx, message1, sizeof(message1));
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
    _exit();
  }

  ret = _sha1_update(&hctx, message2, sizeof(message2));
```

```
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
      _exit();
    }

    ret = _sha1_final(&hctx, digest);
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
      _exit();
    }

    /* Now we get the digest of message1 and message2. */

    _set_response(1, &ret);
    _exit();
  }
```

## 71). _rsa_enc

RSA Encryption.

```
BYTE _rsa_enc(
  BYTE bMode,
  WORD wFid,
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *bMode* | [in] Calculation Mode:<br>• `RSA_CALC_NORMAL` Direct calculation without encoding.<br>• `RSA_CALC_PKCS` Encryption according to PKCS#1 standard. |
| *wFid* | [in] Public file ID |
| *bLen* | [in] Length of the data to be encrypted, different for two modes::<br>• `RSA_CALC_NORMAL` 128 bytes<br>• `RSA_CALC_PKCS` 1~117 bytes |
| *pbData* | [in,out] Plaintext when input, 128 bytes ciphertext when output. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This SES completes the RSA Public key encryption, and the length of the public key is 1024 bits. Different with DES encryption, you have to specify an effective RSA public key file for this function instead of giving the key value directly.

The result after encryption is always 128 bytes long, you have to allocate sufficient storage for pbData.

When using this function for data encryption, setting bMode to be RSA_CALC_PKCS is highly recommended. If using RSA_CALC_NORMAL, try to make the first byte of the 128 bytest data to be 0, otherwise, it may cause the inputted data to be greater than the modulus and this may lead to error. In fact, when inputting data shorter than 128 bytes under RSA_CALC_NORMAL mode, the function will do some padding automatically(add zeroes in the forehead). But, we don’t recommend to input data shorter than 128 bytes. RSA_CALC_NORMAL is commonly used in constructing other algorithm protocols, for instance, digital signature.

There are several ways of generating RSA key pair files in Senselock ELdevice:

- Call API `S4WriteFile()or PS4WriteFile()` to generate a pair of RSA key files in SenseLock EL;
- Call API `S4WriteFile()or PS4WriteFile()` to import a pre-generated RSA key pair into SenseLock EL;
- Using the Devtest.exe in SDK to generate or import a pair of RSA keys to SenseLock EL;

For more detail, please refer to the sample code of corresponding API or relevant instructions for the tool.

RSA keys in Senselock ELare stored using TLV（Tag+Length+Value）format.

The following illustrates the Public key format, 136 bytes in total:

```
typedef struct {
  struct {
    char tag;
    unsigned char length;
    unsigned char value[0x80];
  }n;
  struct {
    char tag;
    unsigned char length;
    unsigned char value[0x04];
  }e;
} COS_RSA_PUBLIC_KEY;
```

Private key can be stored in two formats. One is in quintuple format which takes 330 bytes, is faster, and recommended to use.:

```
typedef struct {
  struct {
    char tag;
    unsigned char length;
    unsigned char value[0x40];
  }p;
  struct {
    char tag;
    unsigned char length;
    unsigned char value[0x40];
  }q;
  struct {
    char tag;
    unsigned char length;
    unsigned char value[0x40];
  }dp;
  struct {
    char tag;
    unsigned char length;
    unsigned char value[0x40];
  }dq;
  struct {
    char tag;
    unsigned char length;
    unsigned char value[0x40];
  }qinv;
} COS_RSA_PRIVATE_KEY;
```

The second one is a simplified format, but it’s slow , not so commonly used and not recommended. It takes 260 bytes totally:

```
typedef struct {
  struct {
    char tag;
    unsigned char length;
    unsigned char value[0x80];
  }n;
  struct {
    char tag;
    unsigned char length;
```

```
        unsigned char value[0x80];
      }d;
    } COS_RSA_PRIVATE_KEY_2;
```

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
please refer to the sample code of _rsa_dec().
```

## 72). _rsa_dec

RSA decryption.

```
BYTE _rsa_dec(
  BYTE bMode,
  WORD wFid,
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *bMode* | [in] Calculation mode:<br>• RSA_CALC_NORMAL Calculation directly without encoding<br>• RSA_CALC_PKCS Decryption according to PKCS#1 standard |
| *wFid* | [in] ID of private key file. |
| *bLen* | [in] Length of the data to be decrypted, must be 128 bytes. |
| *pbData* | [in,out] Ciphertext when input, plaintext when output. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This SES completes the RSA Private key decryption, and the length of the private key is 1024 bits. Different with DES encryption, you have to specify a effective RSA private key file for this function instead of giving the key value directly.

If decrypting using RSA_CALC_NORMAL mode, the result is always 128 bytes long; If using RSA_CALC_PKCS, the result is “data length” + “plaintext”. Namely, first byte of pbData is the length of plaintext, and the following bytes are the decrypted plaintext.

You must adopt the same mode for encryption and decryption, or there will be encoding error.
For the generation and format of RSA keys, please refer to the “Remarks” section of `_rsa_enc()`.

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
/* This demonstrates how to use RSA algorith to encrypt
  some text and and decrypt it later.
  The sample code supposes a pair of valid RSA key files
  have already been created.
*/

#include "ses_v3.h"
#include "string.h"

char plaintext[24] = "This is some test data.";
BYTE ciphertext[128] = "";
BYTE buff[128] = "";
WORD pubkey_fid = 0xc001;
WORD prikey_fid = 0xc002;
```

```
void main()
{
  BYTE ret = 0;

  /* Encrypt… */
  memcpy(ciphertext, plaintext, 24);
  ret = _rsa_enc(RSA_CALC_PKCS, pubkey_fid, 24, ciphertext);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
    _exit();
  }

  /* Decrypt…
    Notice that the ciphertext is 128 bytes long. */
  memcpy(buff, ciphertext, 128);
  ret = _rsa_dec(RSA_CALC_PKCS, prikey_fid, 128, buff);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
    _exit();
  }

  /* Now buff[0] should be length of plaintext, that is, 24.
    And buff[1..25] should be the same as plaintext. */

  _set_response(1, &ret);
  _exit();

}
```

## 73). _rsa_sign

RSA digital signature. There are some limitations for using this SES, for detail, please refer to the “Remarks”section of this table.

```
BYTE _rsa_sign(
  BYTE bMode,
  WORD wFid,
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *bMode* | [in] Signing mode:<br>• `RSA_CALC_NORMAL`<br>Encrypt the HASH value directly using private key;<br>• `RSA_CALC_HASH`<br>Do the HASH operation first for the inputted data and then encrypt the result using private key;<br>• `RSA_CALC_PKCS`<br>Sign the HASH value according to PKCS#1 standard;<br>• `RSA_CALC_HASH|RSA_CALC_PKCS`<br>Do the HASH operation first and then sign the HASH result according to PKCS#1 standard; |
| *wFid* | [in] ID of private key file. |
| *bLen* | [in] The data to be signed, having different requirement for different modes:<br>• `RSA_CALC_NORMAL` 20 bytes<br>• `RSA_CALC_HASH` 1~128 byte(s)<br>• `RSA_CALC_PKCS` 20 bytes<br>• `RSA_CALC_HASH|RSA_CALC_PKCS` 1~128 byte(s) |
| *pbData* | [in,out] Plaintext or HASH value when input, 128 octets signature when output. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This SES completes RSA private key signing operation, and the key length is 1024 bits while the signing result is 128 bytes.

In the four modes above, RSA_CALC_PKCS is a standard digital signing scheme. If you want your signature has a good compatibility with other signing system, you must use this mode. If inputted plaintext are relatively short, you can use `RSA_CALC_HASH|RSA_CALC_PKCS` to do the hashing and signing together. The other two are not recommended.

This SES supports only one HASH algorithm – SHA1. To use other HASH algorithm, use `_rsa_dec()` instead. For a real implemented sample, please refer to the “case study” part of SDK. For the generation and format of RSA keys, please refer to the “Remarks” section of `_rsa_enc()`.

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
please refer to the sample code of_rsa_veri().
```

## 74). _rsa_veri

This SES do RSA digital signature verification according to the HASH buffer and inputted signature. There are some limitations for using this function, and that's why we recommend you to use other means of signature verification demonstrated in "case study" of SDK to replace this SES. For detail, please refer to "Remarks" section of this table.

```
BYTE _rsa_veri(
  BYTE bMode,
  WORD wFid,
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *bMode* | [in] Calculation mode:<br>• `RSA_CALC_NORMAL`<br>Decrypt the signature directly and compare with HASH buffer to verify;<br>• `RSA_CALC_PKCS`<br>According to the Hash buffer and signature, do the standard PKCS#1 verification; |
| *wFid* | [in] ID of public key file. |
| *bLen* | [in] Length of digital signature, must be 128. |
| *pbData* | [in] The signature. |

**Return Values:**

If verified successfully, returns `SES_SUCCESS`. Otherwise, it returns corresponding error code.

**Remarks:**

This SES completes RSA signature verification, and the key length is 1024 bits.

Different with `_rsa_sign()`, this verification function doesn't provide hashing calculation for the plaintext, you have to call corresponding hashing SES to do HASH first and then call this SES. Besides, this SES takes only HASH value from system's HASH buffer and can't take directly inputted HASH value, and that means you can't get the plaintext's HASH value by other ways(i.e. do the hashing in software). If the amount of plaintext is big, there are some inconvenience to handle. In this case, one better way is to use `_rsa_enc()` to implement a verification for the HASH value in your own EXF . For more detail, please refer to the sample code of "case study" in SDK.

This SES supports only one HASH algorithm – SHA1. To use other HASH algorithm, use _rsa_enc() instead. For a real implemented sample, please refer to the "case study" part of SDK. For the generation and format of RSA keys, please refer to the "Remarks" section of _rsa_enc().

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
/* This demonstrates how to make a signature of a short
  message and then verify it later.
  The sample code supposes a pair of valid RSA key files
  have already been created.
*/

#include “ses_v3.h”
#include “string.h”

char message[24] = “This is some test data.”;
BYTE signature[128] = “”;
BYTE digest[20];
WORD pubkey_fid = 0xc001;
WORD prikey_fid = 0xc002;
HASH_CONTEXT hctx;

void main()
{
  BYTE ret = 0;

  /* Make signature… */
  memcpy(signature, message, 24);
  ret = _rsa_sign(RSA_CALC_HASH|RSA_CALC_PKCS, prikey_fid,
          24, signature);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
    _exit();
  }

  /* Verify… */
  /* Hash the message first… */
  _sha1_init(&hctx);
  _sha1_update(&hctx, message, 24);
  _sha1_final(&hctx, digest);

  /* Start to verify… */
  ret = _rsa_veri(RSA_CALC_PKCS, pubkey_fid, 128, signature);
  if (ret != SES_SUCCESS)
  {
    _set_response(1, &ret);
    _exit();
  }

  /*  ret = SES_SUCCESS means the signature is valid. */

  _set_response(1, &ret);
  _exit();

}
```

## 75). _rand

Hardware generated real random number.

```
BYTE _rand(
  BYTE bLen,
  BYTE *pbData);
```

**Parameters:**

| | |
|---|---|
| *bLen* | [in] Length of the random number to be generated, 1~255. |
| *pbData* | [out] Random number generated. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Random number produced by this SES is a hardware generated real one and needs no initialization..

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
/* This demonstrates how to generate 128 bytes of random
  data and returned it to PC.
*/

#include “ses_v3.h”

/* Customized IO package encoding structure. */
typedef struct {
    /* Flag may be used for error-code or fucntion code. */
  BYTE flag;
  /* Data length in buff. */
  BYTE len;
  /* Data transmitted in or out. */
  BYTE buff[1];
} IO_PACKAGE;

unsigned char tmp[128+2];
IO_PACKAGE *out = (IO_PACKAGE *)tmp;

void main()
{
  BYTE ret = 0;

  /* Generate random data… */
  ret = _rand(128, out->buff);
  if (ret != SES_SUCCESS)
  {
    out->flag = ret;
    out->len = 0;
    _set_response(2+out->len, (BYTE *)out);
    _exit();

  }
```

```
      /* Return random data… */
      out->flag = ret;
      out->len = 128;
      _set_response(2+out->len, (BYTE *)out);
      _exit();

    }
```

## 8.6. Memory Operation

Functions of this category provide basic memory operations, i.e. memory copy, memory move etc. They are offered for the same purpose as single precision float function: to enhance the efficiency of code execution and decrease the size of objective executable. If one doesn't care about the efficiency of execution and the size of objective executable, he can use those functions defined in *string.h* , for example, `memcpy()`.

### 76). _mem_copy

Copy data between memory buffers, similar as `memcpy()` in standard C.

```
BYTE _mem_copy(
  void *pDest,
  void *pSrc,
  BYTE bLen);
```

**Parameters:**

| | |
|---|---|
| *pDest* | [in] Destination address. |
| *pSrc* | [in] Source address. |
| *bLen* | [in] Length of the data to be copied. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Different with `memcpy` in standard C, `_mem_copy` returns SES error code instead of a pointer to destination address.

**Requirements:**

Hardware version:Senselock EL2.x

## 77). _mem_move

Move data from one buffer to another. Overlay of destination buffer and source buffer is handled, similar as `memmove()` in standard C.

```
BYTE _mem_move(
  void *pDest,
  void *pSrc,
  BYTE bLen);
```

**Parameters:**

| | |
|---|---|
| *pDest* | [in] Destination address. |
| *pSrc* | [in] Source address. |
| *bLen* | [in] Length of the data to be moved. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Different with `memmove` in standard C, `_mem_move` returns SES error code instead of a pointer to destination address.

**Requirements:**

Hardware version:Senselock EL2.x

## 78). _mem_set

Fill the buffer with the value specified, similar as `memset()` in standard C.

```
BYTE _mem_set(
  void *pDest,
  BYTE c,
  BYTE bLen);
```

**Parameters:**

| | |
|---|---|
| *pDest* | [in] Destination address. |
| *c* | [in] Value to be filled in. |
| *bLen* | [in] Length of the buffer to be filled. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Different with `memset` in standard C, `_mem_set` returns SES error code instead of a pointer to destination address.

**Requirements:**

Hardware version:Senselock EL2.x

## 79). _mempool_init

Initialize the starting address and size of memory pool, used for dynamic memory management. It must be called before any one of `_malloc()`,`_calloc()`or `_realloc()`.

```
BYTE _mempool_init(
  void *pStart,
  WORD wSize);
```

**Parameters:**

| | |
|---|---|
| *pStart* | [in] Starting address of memory pool, must be a pointer to XRAM. |
| *wSize* | [in] Size of the memory pool. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

The internal resources in Senselock ELcan't be used as freely as in PC. Try to use static memory instead of dynamic means unless you are very familiar with memory structure and usage.

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
_mempool_init((void xdata*)0x0400, 1024);
```

## 80). _malloc

Malloc a memory block in the memory pool set by `_mempool_init()`.

```
void* _malloc(WORD wSize);
```

**Parameters:**

| | |
|---|---|
| *wSize* | [in] Size of the memory to be malloced. |

**Return Values:**

Return pointer to the starting address of the memory block malloced if succeed, NULL otherwise.

**Remarks:**

Must call `_mempool_init()` first to set an available memory pool.

The internal resources in Senselock ELcan't be used as freely as in PC. Try to use static memory instead of dynamic means unless you are very familiar with memory structure and usage.

**Requirements:**

Hardware version:Senselock EL2.x

## 81). _calloc

Allocate an array in the memory pool set by `_mempool_init()` with elements initialized to 0.

```
void* _calloc(
  WORD wNobj,
  WORD wSize);
```

**Parameters:**

| | |
|---|---|
| *wNobj* | [in] Number of the elements. |
| *wSize* | [in] Length in bytes of each element. |

**Return Values:**

Return pointer to the starting address of the memory block malloced if succeed, NULL otherwise.

**Remarks:**

Must call _mempool_init() first to set an available memory pool.

The internal resources in Senselock ELcan't be used as freely as in PC. Try to use static memory instead of dynamic means unless you are very familiar with memory structure and usage.

**Requirements:**

Hardware version:Senselock EL2.x

## 82). _realloc

Reallocate memory blocks in the memory pool set by `_mempool_init()`.

```
void* _realloc(
  void *pointer,
  WORD wSize);
```

**Parameters:**

| | |
|---|---|
| *pointer* | [in] Pointer to previously allocated memory block. |
| *wSize* | [in] New size in bytes. |

**Return Values:**

Return pointer to the starting address of the memory block malloced if succeed, NULL otherwise.

**Remarks:**

Must call `_mempool_init()` first to set an available memory pool.

The internal resources in Senselock ELcan’t be used as freely as in PC. Try to use static memory instead of dynamic means unless you are very familiar with memory structure and usage.

**Requirements:**

Hardware version:Senselock EL2.x

## 83). _free

Free allocated memory block.

```
BYTE _free(void *pointer);
```

**Parameters:**

| | |
|---|---|
| *pointer* | [in] Pointer to the memory block to be freed. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 84). _invert

Invert the content of memory buffer.

```
BYTE _invert(
  void *pvdata,
  BYTE bLen);
```

**Parameters:**

| | |
|---|---|
| *pvdata* | [in] Pointer to the memory buffer. |
| *bLen* | [in] Length of the data to be inverted. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This SES can invert the higher and lower part of a specified memory buffer, used for the conversion between data of Big-Endian and Little-Endian.
For example, data in a memory block are: 0x12, 0x34, 0x56, 0x78. After calling _invert(), they become to be: 0x78 0x56 0x34 0x12.

**Requirements:**

Hardware version:Senselock EL2.x

## 85). _mem_cmp

Compare characters in two buffers.

```
char _mem_cmp(
  void *pdest,
  void *psrc,
  BYTE length);
```

**Parameters:**

| | |
|---|---|
| *pdest* | [in] First buffer. |
| *psrc* | [in] Second buffer. |
| *bLen* | [in] Number of characters. |

**Return Values:**

Return Values indicates the relationship between the buffers:

- `< 0` *pdest* less than *psrc*
- `= 0` *pdest* identical to *psrc*
- `> 0` *pdest* greater than *psrc*

**Remarks:**

The `_mem_cmp` function compares the first *bLen* bytes of *pdest* and *psrc* and returns a value indicating their relationship.

**Requirements:**

Hardware version: Senselock EL2.3

## 8.7. Time

Two time functions are provided in Senselock ELhardware: Timer and Real Time Clock. The former is a standard function for all Senselock ELdevice, and the latter is supported only for specific version of Senselock ELwith clock function.

Timer is substantively a 64 bits counter of CPU. You can use the current reading of the counter and the frequency of CPU to calculate the real elapsed time. Timer can be used only when Senselock ELis working properly. On the other hand, Real Time Clock is powered by independent battery, it can work finely even when Senselock ELis unplugged from computer.

### 86). _set_timer

Set timer's initial value and working mode.

```
BYTE _set_timer(
  BYTE bMode,
  DWORD  *pdwCount);
```

**Parameters:**

| | |
|---|---|
| *bMode* | [in] Timer mode:<br>• 0 Non-cycle mode, the timer stops when the counter number overflows<br>• 1 Cycle mode, the timer restarts when the counter number overflows<br>• 2 Cycle reset mode, the timer restarts from initial value after the counter number overflows |
| *pdwCount* | [in] Pointer to the initial value. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

The timer uses counting-up mechanism.

**Requirements:**

Hardware version:Senselock EL2.x

## 87). _start_timer

Start the timer according to the specified working mode and initial value.

```
BYTE _start_timer();
```

**Parameters:**

*None.*

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 88). _stop_timer

Stop the timer.

```
BYTE _stop_timer();
```

**Parameters:**

*None.*

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

None.

**Requirements:**

Hardware version:Senselock EL2.x

## 89). _get_timer

Get current reading of the timer.

```
BYTE _get_timer(DWORD *pdwCount);
```

**Parameters:**

| | |
|---|---|
| *pdwCount* | [out] Address of the variable for storing current reading of the timer. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

The timer uses counting up mechanism, and the counter increases itself by one for 64 CPU clock cycles. The CPU frequency, of before and including v2.3.2 EL is 16MHZ; of after v2.3.2 EL is 18MHZ. The time span represented by one counter unit is:

```
1*64/16000000 = 4μs (before and including
v2.3.2)
1*64/18000000 = 3.6μs (after v2.3.2)
```

The counter number is stored in a DWORD variable, so the total time can be represented by the counter before its overflow is:

```
  0xffffffff*4/1000000/3600 ≈ 4.77 hours (before and including
v2.3.2)
  0xffffffff*3.6/1000000/3600 ≈ 4.29 hours (after v2.3.2)
```

It is available to use development test tool, or device check tool to get the device version number, or go to the struct SENSE4 CONTEXT's filed to get. When the hardware version of EL is 2.3.4, the dsVersion is 0x00020304. Note: the SENSE4 CONTEXT must be enumerated successfully. About the SENSE4 CONTEXT, please refer to the note of S4Enum in 9.1.1.

**Requirements:**

Hardware version:Senselock EL2.x

**Example Code:**

```
/* This demonstrate how to use a Timer to determine the running
time of a functon. */
#include "ses_v3.h"

DWORD tick_count = 0;
float time_used = 0;

void foo()
{
  /* Operations here…*/
}

void main()
{
  BYTE ret = 0;

  tick_count = 0;
  ret = _set_timer(1, &tick_count);
```

```
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
      _exit();
    }

    ret = _start_timer();
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
      _exit();
    }

    foo();

    ret = _get_timer(&tick_count);
    if (ret != SES_SUCCESS)
    {
      _set_response(1, &ret);
      _exit();
    }

    /* Get time used in millisecond. */
    time_used = (tick_count - 0) * 4/1000;

    _stop_timer();

    _set_response(1, &ret);
    _exit();

  }
```

## 90). _time

Get current time, returning the number of seconds since midnight(00:00:00), January 1, 1970, coordinated universal time. Similar as `time()` in standard C.

```
BYTE _time(time_t *ptime);
```

**Parameters:**

| | |
|---|---|
| *Ptime* | [out] Address of the variable for holding the current time. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

The self-powered clock returns time of GMT format. If there is time difference problem, one may need to adjust it according to local time zone.

If there is no suffient battery power, even after you may replace it with a new battery, this function will still return error. Initialization after battery replacement can only be done by Senselock ELmanufacturer.

Different with `time()` in standard C, `_time()` returns SES error code instead of a time value. So please make sure the effectiveness of input pointer parameter(for returning data).

**Requirements:**

Hardware version: Senselock EL2.3 with clock function.

**Example Code:**

```
please refer to the sample code of_gmtime().
```

## 91). _mktime

Converts the local time(time structure) to a calendar value(time_t).

```
BYTE _mktime(
  time_t    *ptime,
  RTC_TIME_T  *ptm);
```

**Parameters:**

| | |
|---|---|
| *ptime* | [out] Pointer to the converted time_t. |
| *ptm* | [in] Pointer to a RTC_TIME_T struct, please refer to "Remarks" section. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

RTC_TIME_T struct definition:

```
typedef struct _RTC_TIME_T {
  BYTE second; /* second, 0~59 */
  BYTE minute; /* minute, 0~59. */
  BYTE hour;    /* hour, 0~24. */
  BYTE day;     /* day of month, 1~31. */
  BYTE week;    /* day of the week, Sunday is 0, Monday
                    is 1, and so on. */
  BYTE month;   /* month, 0~11,  January is 0 */
  WORD year;    /* year (100- 138, Year 2000 is 100. */
} RTC_TIME_T;
```

When convert from RTC_TIME_T to time_t, member variable *week* is ignored. Different with mktime() in standard C, _mktime() doesn't return time_t. It returns SES error code instead. Parameter *ptime* then is used to return the value of time_t.

**Requirements:**

Hardware version: Senselock EL2.3 with clock function.

**Example Code:**

```
please refer to the sample code of_gmtime().
```

## 92). _gmtime

Convert a `time_t` value to `RTC_TIME_T` struct.

```
BYTE _gmtime(
  time_t     *ptime,
  RTC_TIME_T  *ptm);
```

**Parameters:**

| | |
|---|---|
| *ptime* | [in] Pointer to `time_t` variable. |
| *ptm* | [out] Pointer to `RTC_TIME_T` strucrt, please refer to the “Remarks” section of this table. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This function is for the convenience of handling time in Senselock ELinternally. It request that the year represented by `time_t` is not earlier than A.D.2000, or it will lead to error.
Different with `gmtime()` in standard C, `_gmtime()` returns SES error code. Parameter *ptm* is used to return the `RTC_TIME_T` struct.

**Requirements:**

Hardware version: Senselock EL2.3 with clock function.

**Example Code:**

```
/* This demonstrate how to set an expiration date on specific
function */

#include “ses_v3.h”

#define MY_SUCCESS 0x00
#define MY_ERROR_SES 0x01
#define MY_ERROR_EXPIRED 0x02

RTC_TIME_T exp = {0,0,0,20,0,11,105};/* 2005.12.20 */
time_t et;

BYTE foo()
{
  BYTE ret = 0;
  time_t t = 0;

  /* Get current time. */
  ret = _time(&t);
  if (ret != SES_SUCCESS)
  {
    return MY_ERROR_SES;
  }

  /* Convert from RTC_TIME_T structure. */
  ret = mktime(&et, &exp);
  if (ret != SES_SUCCESS)
  {
    return MY_ERROR_SES;
  }

  /* Compare to expiration date. */
  if (t > et)
  {
    /* if you want to display current time, you can convert
    it to RTC_TIME_T structure. For example:
    RTC_TIME_T cur;
    …
    ret = _gmtime(&t, &cur);
    _set_response(sizeof(RTC_TIME_T), (BYTE *)&cur);
    _exit();
    */
    return MY_ERROR_EXPIRED;
  }

  /* do something here. */

  return MY_SUCCESS;
}

void main()
{
  foo();

  _exit();
}
```

## 8.8. Macro and Auxiliary function

For convenience of programming, Senselock ELprovides some additional macros and auxiliary functions which are implemented by coding instead of being offered by Senselock ELoperating system.

### 93). _swap_u16

This Macro is used to invert a 2 bytes memory.

```
_swap_u16(__x__);
```

**Parameters:**

| | |
|---|---|
| *__x__* | [in] Address of a 2 bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro is kept for compatibility purpose only, try to use `invert()` instead. The latter is defined as follows:

```
#define _swap_u16(__x__) _invert(__x__, 2)
```

**Requirements:**

Hardware version:Senselock EL2.x

## 94). _swap_u32

This Macro is used to invert a 4 bytes memory.

```
_swap_u32(__x__);
```

**Parameters:**

| | |
|---|---|
| *__x__* | [in] Address of a 4 bytes variable.. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro is kept for compatibility purpose only, try to use `invert()` instead. The latter is defined as follows:

```
#define _swap_u32(__x__) _invert(__x__, 4)
```

**Requirements:**

Hardware version:Senselock EL2.x

## 95). LE16_TO_CC

This macro converts a Little-Endian 2 bytes variable(i.e. `short`) to a byte sequence supported by current compiler.

```
LE16_TO_CC(__x__);
```

**Parameters:**

| | |
|---|---|
| *__x__* | [in] Address of a 2 bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro can convert byte sequence automatically according to current compiler type.One can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other two compilers use Little-Endian, this macro is equal to `_swap_u16()` when used in Keil C51, and it does nothing when used in other two compilers.

**Requirements:**

Hardware version:Senselock EL2.x

## 96). LE32_TO_CC

This macro converts a Little-Endian 4 bytes variable(i.e. `long`) to a byte sequence supported by current compiler.

```
LE32_TO_CC(__x__);
```

**Parameters:**

| | |
|---|---|
| *__x__* | [in] Address of a 4 bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro can convert byte sequence automatically according to current compiler type.One can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other two compilers use Little-Endian, this macro is equal to `_swap_u32()` when used in Keil C51, and it does nothing when used in other two compilers.

**Requirements:**

Hardware version:Senselock EL2.x

## 97). CC_TO_LE16

This macro, contrary to LE16_TO_CC, convert a 2 bytes variable to Little-Endian according to compiler type.

```
CC_TO_LE16(__x__);
```

**Parameters:**

| | |
|---|---|
| __*x*__ | [in] Address of a two bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro can convert byte sequence automatically according to current compiler type.One can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other two compilers use Little-Endian, this macro is equal to `_swap_u16()` when used in Keil C51, and it does nothing when used in other two compilers.

**Requirements:**

Hardware version:Senselock EL2.x

## 98). CC_TO_LE32

This macro, contrary to LE32_TO_CC, convert a 4 bytes variable to Little-Endian according to compiler type.

```
CC_TO_LE32(__x__);
```

**Parameters:**

| | |
|---|---|
| *__x__* | [in] Address of a 4 bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro can convert byte sequence automatically according to current compiler type.One can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other two compilers use Little-Endian, this macro is equal to `_swap_u32()` when used in Keil C51, and it does nothing when used in other two compilers.

**Requirements:**

Hardware version:Senselock EL2.x

## 99). BE16_TO_CC

This macro converts a Big-Endian 2 bytes variable(i.e. `short`) to a byte sequence supported by current compiler.

```
BE16_TO_CC(__x__);
```

**Parameters:**

| | |
|---|---|
| *__x__* | [in] Address of a 2 bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro can convert byte sequence automatically according to current compiler type.One can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other two compilers use Little-Endian, this macro does nothing when used in Keil C51, and it is equal to `_swap_u16()` when used in other two compilers.

**Requirements:**

Hardware version:Senselock EL2.x

## 100). BE32_TO_CC

This macro converts a Big-Endian 4 bytes variable(i.e. `long`) to a byte sequence supported by current compiler.

```
BE32_TO_CC(__x__);
```

**Parameters:**

| | |
|---|---|
| __*x*__ | [in] Address of a 4 bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro can convert byte sequence automatically according to current compiler type.One can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other two compilers use Little-Endian, this macro does nothing when used in Keil C51, and it is equal to `_swap_u32()` when used in other two compilers.

**Requirements:**

Hardware version:Senselock EL2.x

## 101). CC_TO_BE16

This macro, contrary to BE16_TO_CC, convert a 2 bytes variable to Little-Endian according to compiler type.

```
CC_TO_BE16(__x__);
```

**Parameters:**

| | |
|---|---|
| *__x__* | [in] Address of a 2 bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro can convert byte sequence automatically according to current compiler type.One can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other two compilers use Little-Endian, this macro does nothing when used in Keil C51, and it is equal to `_swap_u16()` when used in other two compilers.

**Requirements:**

Hardware version:Senselock EL2.x

## 102). CC_TO_BE32

This macro, contrary to BE32_TO_CC, convert a 4 bytes variable to Little-Endian according to compiler type.

```
CC_TO_BE32(__x__);
```

**Parameters:**

| | |
|---|---|
| *__x__* | [in] Address of a 4 bytes variable. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This macro can convert byte sequence automatically according to current compiler type.One can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other two compilers use Little-Endian, this macro does nothing when used in Keil C51, and it is equal to `_swap_u32()` when used in other two compilers.

**Requirements:**

Hardware version:Senselock EL2.x

## 103). _atod

Convert a string to double precision float.

```
BYTE _atod(
  DOUBLE_T *presult,
  char   *pstr);
```

**Parameters:**

| | |
|---|---|
| *presult* | [out] Pointer to the double precision float struct. |
| *Pstr* | [in] String representation of a float, for example “3.1415926”. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

This function is provided with source code. Its header file and source code are *atod.h* and *atod.c*, please search those files under the same directory as *ses_v3.h*
Using this function will consume much system resource. If there is no too strict requirement for the initial value, try to use other system functions to assign a `double` float, for instance , `_ftod()`, or to initialize memory storage directly.

This function used to be named `_strtod()`, and is changed to be `_atod()` due to naming conflict in new version compiler.

**Requirements:**

Hardware version: Senselock EL2.3

## 104). DEFINE_AT

Define a variable at a specified address according to compiler type.

```
DEFINE_AT(TYPE, NAME, ADDRESS, MEMORY)
```

**Parameters:**

| | |
|---|---|
| *TYPE* | [in] Type of the variable to be defined. |
| *NAME* | [in] Variable name. |
| *ADDRESS* | [in] Address of the variable specified. |
| *MEMORY* | [in] memory zone:<br>• RAM_EXT xdata<br>• RAM_INT_DE data<br>• RAM_INT_ID idata<br>• ROM code memory |

**Return Values:**

None..

**Remarks:**

None..

**Requirements:**

Hardware version:Senselock EL2.x

## 8.9. Get Device Info

A SES which can read basic hardware information is added in Senselock EL2.3.2. It can read the GUSN(Global Unique Serial Number) and developer ID (also called: customer ID).

### 105). _get_gbdata

Read GUSN or developer ID.

```
BYTE  _get_gbdata(
  BYTE bFlag,
  BYTE* pbData,
  BYTE bLen);
```

**Parameters:**

|  |  |
|---|---|
| *bFlag* | [in] Flag:<br>• GLOBAL_SERIAL_NUMBER get the 8 octets GUSN<br>• GLOBAL_CLIENT_NUMBER get the 4 octets customer ID |
| *pbData* | [out] Buffer for holding data read. |
| *bLen* | [in] size of the buffer *pbData*. |

**Return Values:**

Return SES_SUCCESS if succeed or corresponding error code otherwise.

**Remarks:**

Parameters *bLen* is used to represent the size of *pbData*. It is not necessary for its value to be big enough for holding all the data. For example, if you specify *bLen* =5 when reading hardware GUSN(8 octets), this function gets only the formost 5 bytes of GUSN and returns no error.

**Requirements:**

Hardware version: Senselock EL2.3.2
Compiler-dependent: Support only Skit and Keil, Do NOT support Rkit

## 8.10. SES Error Code List

| Error code | Value | Comments |
|---|---|---|
| SES_SUCCESS | 0x00 | Succeed |
| SES_MSC | 0x01 | Unsupported SES invoking |
| SES_PARA | 0x02 | Invalid parameter |
| SES_EEPROM | 0x03 | Error writing EEPROM |
| SES_RAM | 0x04 | Memory violation |
| SES_XCOS | 0x05 | Error from XCOS |
| SES_FILEID | 0x11 | Incorrect file ID, file not found |
| SES_FILE_ACCESS | 0x12 | Error accessing file |
| SES_FILE_SELECT | 0x13 | Error selecting file |
| SES_HANDLE | 0x14 | Invalid file handle |
| SES_RANGE | 0x15 | File out of range |
| SES_FILE_SPACE | 0x16 | Insufficient file storage |
| SES_FILE_EXISTING | 0x17 | File already exists |
| SES_KEYID | 0x21 | Error Key file ID |
| SES_KEY_ACCESS | 0x22 | Access to key file failed |
| SES_SHA1 | 0x23 | SHA1 calculation error |
| SES_RAND | 0x24 | Error getting random number |
| SES_RSA | 0x25 | RSA calculation error |
| SES_RSAVERIFY | 0x26 | RSA verification failed, incorrect signature |
| SES_INVALID_POINTER | 0x30 | Invalid memory pointer |
| SES_INVALID_SIZE | 0x31 | Invalid memory size |
| SES_REAL_TIME | 0x40 | Error reading real time |

## 8.11. Double Precision Float Boundary Limitation

Under some marginal condition, double precision float calculation may have bias.

For functions marked with # in following table, the absolute value for inputting parameter can't be less than $1\times10^{-154}$, (exclude 0), or it will exceed the boundary of float representable by SenseLock EL. When the result exeeds the representable boundary, Return Value(presult) will NOT indicate being out of range.

| Function protocol | Boundary Condition Clarification |
|---|---|
| `BYTE _mult( DOUBLE_T* presult, DOUBLE_T* px, DOUBLE_T* py)` | # |
| `BYTE _sin( DOUBLE_T* presult, DOUBLE_T* px)` | #When the value of *px is close to multiple of π/2, there will be relatively big bias in result. When it is bigger than $1\times10^{5}$时,the calculation precision will decrease to 9 effective digits due to accumulated bias. |
| `BYTE _cos( DOUBLE_T* presult, DOUBLE_T* px)` | #When the value of *px is close to multiple of π/2, there will be relatively big bias in result. When it is bigger than $1\times10^{5}$时,the calculation precision will decrease to 9 effective digits due to accumulated bias. |
| `BYTE _tan( DOUBLE_T* presult, DOUBLE_T* px)` | #When the value of *px is close to multiple of π/2, there will be relatively big bias in result. When it is bigger than $1\times10^{5}$时,the calculation precision will decrease to 9 effective digits due to accumulated bias. |
| `BYTE _asin( DOUBLE_T* presult, DOUBLE_T* px)` | #When the value of *px exeeds the domain of ±1 , Return Values indicates NO error. |
| `BYTE _acos( DOUBLE_T* presult, DOUBLE_T* px)` | #When the value of *px exeeds the domain of ±1 , Return Values indicates NO error. |
| `BYTE _atan( DOUBLE_T* presult, DOUBLE_T* px)` | # |
| `BYTE _sinh( DOUBLE_T* presult, DOUBLE_T* px)` | When the absolute value of *px is less than $1\times10^{-8}$ (exclude 0), the calculation precision will decrease to 9 effective digits. The value of *px can NOT be greater than 709. |
| `BYTE _cosh( DOUBLE_T* presult, DOUBLE_T* px)` | When the absolute value of *px is less than $1\times10^{-8}$ (exclude 0), the calculation precision will decrease to 9 effective digits. The value of *px can NOT be greater than 709. |
| `BYTE _tanh( DOUBLE_T* presult, DOUBLE_T* px)` | When the absolute value of *px is less than $1\times10^{-8}$ (exclude 0), the calculation precision will decrease to 9 effective digits. The value of *px can NOT be greater than 709. |
| `BYTE _atan2( DOUBLE_T* presult, DOUBLE_T* px, DOUBLE_T* py)` | #The absolute value of *px÷*py must between $1\times10^{-154}$and 0（exclude 0） |
| `BYTE _fmod( DOUBLE_T* presult,` | There will be relatively big bias when the value of *px÷*py is larger than $10^{6}$ |

| | |
|---|---|
| `DOUBLE_T* px, DOUBLE_T* py)` | |
| `BYTE _exp( DOUBLE_T* presult, DOUBLE_T* px)` | The value of *px can’t bigger than 709 |
| `BYTE _pow( DOUBLE_T* presult, DOUBLE_T* px, DOUBLE_T* py)` | Even when the scale of result is less than the representable boundary of double, it will NOT be set to zero. |
| `BYTE _modf( DOUBLE_T* presult, DOUBLE_T* px, DOUBLE_T* intptr)` | The absolute value of *px must between $1\times10^{18}$ and $1\times10^{-23}$. (include 0) |
| `BYTE _ldexp( DOUBLE_T* presult, DOUBLE_T* px, int exp)` | Even when the scale of result is less than the representable boundary of double, it will NOT be set to zero. |
| `BYTE _dtof( float* presult, DOUBLE_T* px)` | Even when *px is bigger than the representable boundary of double, return value indicates NO overflow. Even when *px is less than the representable boundary of double, it will NOT be set to zero. |
| `BYTE _dtol( long* presult, DOUBLE_T* px)` | Even when *px is bigger than the representable boundary of long, return value indicates NO overflow. Even when *px is less than the representable boundary of long, it will NOT be set to zero. |

# 9. API Reference

Senselock ELprovides two groups of API functions, used for local and network operations respectively. Local APIs are used for Senselock ELdesktop version, and for the local operations (initialization and development) of network version; Network APIs are used for accessing Senselock ELof network version remotely.

Local API is sense4.dll, Network API is sense4user.dll, both of them are offed with static lib as well.

Please note:

As the v3.0 network dongle has different file system from the previous, the device being created by the previous API or tool, are only accessable to previous API or tool and vice versa, otherwise, the communication error will come out. About the detail on network dongle, please refer to the 7th Chapter Manual on Network Dongle, for standalone version, there is no such restrictions.

All network dongle refered in this guide, unless being specific noted, is pointing to v3.0 API, tool, and network device created by v3.0 API.

The following table illustrates the relationship between software API and the dongle.

Table 8-1

| | Desktop version SenseLock EL | Network version SenseLock EL |
|---|---|---|
| Local API | Applicable | Applicable, but limited to local operations such as initialization etc only. |
| Network API | Not applicable | Applicable, used for accessing Senselock ELdevice remotely in software. |

Senselock ELSDK provides APIs of various language versions, i.e. C, Delphi, Java ect. According to the programming language used in your software, you can choose corresponding API library. To those languages not supported in SDK, you can use dll (dynamic link library) instead.

For better result in software protection, it’s recommended to use static library of these programming languages. For better compatibility and more convenience in software upgrading, dll (dynamic link library) is recommended.

In the chapter of “Develop Senselock ELcodes”, we make some basic introduction to the usage of API functions. This chapter, will further focus on the detail of these API functions and on what attentions should be paid when using them. If one hopes to understand the general procedure of accessing Senselock ELby API, please refer to the fourth chapter “Develop Senselock ELcodes”.

Some APIs are used only in the stage of developing dongle applications, while some other APIs are used for accessing the dongle in the end user environment. **NEVER use API functions marked as “developer only” in the client end as shown in the following table.**

Table 8-1 Illustration of the conditions for using API functions

| API Function | Remarks | Using Conditions | Supportive API |
|---|---|---|---|
| `S4Enum` | List all the devices currently connected | Developer, user | Local & Network |
| `S4Open` | Connect the device with specified index | Developer, user | Local & Network |
| `S4OpenEx` | Connect the device specified using specific mode | Developer, user | Local |
| `S4Close` | Close connection to the device specified | Developer, user | Local & Network |
| `S4Control` | Send control codes to the device, for example, the lighting control of LED | Developer, user | Local & Network[13] |
| `S4CreateDir` | Create a new directory under current directory, it's recommended to use `S4CreateDirEx` instead. | Developer only | Local |
| `S4CreateDirEx` | Create a new directory under current directory | Developer only | Local |
| `S4ChangeDir` | Change current working directory | Developer, user | Local & Network |
| `S4EraseDir` | Erase all the files and sub directories under the directory specified. | Developer only | Local |
| `S4VerifyPin` | Verify PIN code | Developer, user | Local & Network |
| `S4ChangePin` | Change PIN code | Developer only[14] | Local |
| `S4WriteFile` | File creation, update; RSA key pair generation, setup authorization | User only | Local |
| `S4Execute` | Execute VM EXF | User only | Local & Network |
| `S4ExecuteEx` | Execute VM EXF or XA EXF | User only | Local |
| `PS4WriteFile` | Similar to `S4WriteFile`,can operate disk file directly | Developer only | Local |
| `S4Startup` | Revoked | — | — |
| `S4Clearup` | Revoked | — | — |

## 9.1. API List

This section will make detailed introduction to all Senselock ELAPIs. To each API, we will illustrate its function from two aspects: "local" and "network".

[13] Local and Network API supports different Control Code, please check out note of S4Control.
[14] If it's to change user PIN, one can also use this function in the client end.

### 9.1.1. S4Enum

Local: List all the available Senselock ELdevices currently connrected to the local computer, and store the device information obtained to `S4CtxList`.
Network: List all the software protection services on network version server and store the obtained information to `S4CtxList`.

```
DWORD WINAPI S4Enum(
  SENSE4_CONTEXT *pS4CtxList,
  DWORD *pdwSize);
```

**Parameters:**

| | |
|---|---|
| *pS4CtxList* | [out] Pointer to `SENSE4_CONTEXT` struct array. |
| *pdwSize* | [in,out] The size of `SENSE4_CONTEXT` struct array(in byte) when input, length of the data actually returned when output. If `pS4CtxList=NULL` or the size of the provided struct array is insufficient, the function will return error and save the size of the storage actually needed to the variable specified by *pdwSize*. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Before calling any other functions, one must call this function to obtain device information. Information for each device is stored in a single struct `SENSE4_CONTEXT` which is defined as follows:

```
  typedef struct {
    DWORD    dwIndex;
    DWORD    dwVersion;
    S4HANDLE hLock;
    BYTE   reserve[12];
    BYTE   bAtr[MAX_ATR_LEN];
    BYTE   bID[MAX_ID_LEN];
    DWORD    dwAtrLen;
  } SENSE4_CONTEXT, *PSENSE4_CONTEXT;
```

Explanation for each member variable of this struct follows below:

| | |
|---|---|
| *dwIndex* | Index for current device, begin with 0. |
| *dwVersion* | Senselock ELhardware version number, currently support:<br>• `SENSE4_CARD_TYPE_V2_00` version 2.0<br>• `SENSE4_CARD_TYPE_V2_01` version 2.1<br>• `SENSE4_CARD_TYPE_V2_02` version 2.2<br>• `SENSE4_CARD_TYPE_V2_30` version 2.3 |
| *hLock* | Device Handle. |
| *reserve* | System reserved. |
| *bAtr* | ATR information, system reserved. |
| *bID* | Device ID. If there is an ATR file in the hardware, then *bID* will hold a self defined ID(content of the ATR file), otherwise, it will hold the Hardware Unique Serial Number(HUSN) of the hardware. For detail information about ATR file, please refer to the Remark section of `S4CreateDirEx()`.For network API, *bID* will hold the ID of the master key on network server upon function |

| | |
|---|---|
| | return. |
| *dwAtrLen* | Length of ATR, system reserved. |

If you doesn't know previously the size of `SENSE4_CONTEXT` struct array needed, you can set the input parameter *pS4CtxList* to be `NULL`, then the parameter *pdwSize* will return the memory storage actually needed. If the system does not connect Elite EL, parameter pdwSize will return 0, and S4_SUCCESS.
The number of Senselock ELconnected in current system can be calculated using the value returned by parameter *pdwSize* as follows:

```
Num = *pdwSize/sizeof(SENSE4_CONTEXT)
```

To network version, it lists available network services instead of real devices. Because network service and local Senselock ELdevice are very similar in real usage, we directly call `SENSE4_CONTEXT` to be "device information" for the sake of convenience.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

**Sample:**

Please refer to the section 4.2.1 "General access".

### 9.1.2. S4Open

Local: Connect to the Senselock ELdevice specified. Connection must be established before any access can be made.
Network: Obtain the connection to the network version device server.

```
DWORD WINAPI S4Open(SENSE4_CONTEXT *pS4Ctx);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in,out] Pointer to the `SENSE4_CONTEXT` struct of the Senselock ELdevice to be opened. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Use struct `SENSE4_CONTEXT` returned from `S4Enum()` as input parameter to establish a connection to this device.

To local API, this function connects to the device using share mode, namely, device connected supports multiple-process access; To network API, there is no differentiation for exclusive mode and share mode. For details regarding exclusive mode and share mode, please refer to the “Remarks” section of `S4OpenEx()`.

If there are multiple Senselock ELdevices in local system, it is very important to assure that correct device is opened. Two important reasons: one is that if connected to the wrong device, the software will mistakenly think that the dongle is illegal and thus result in the software execution error; the other one is that if have operated the dongle of another software, it will change the status of that dongle and thus infulence the normal execution of that software. To avoid these problems and open correct dongle, one can take the following two commonly used approaches:

The first one, create and manage your own device ID.
This is the most effective method. When creating the root directory of a dongle, one can create a self-defined ATR file, content of which contains self-defined 8 octets device ID. When running `S4Enum()`to list devices, IDs of all the dongles will be stored in the *bID* member variables of `SENSE4_CONTEXT` struct array, and thus you can distinguish different devices according to different IDs. For the details regarding how to create ATR file, please refer to the Remarks section of `S4CreateDirEx()`.

The second one, use dongle’s “Developer ID” (also called “customer ID”).
Senselock ELoffers the service of customizing “Developer ID”, which means Senselock can allocate different “Developer ID” for different software vendors. This ID has no relation with HUSN or self-defined device ID and it can be acquired by `S4Control()`. Software can distinguish different devices according to different “Developer ID”. For details regarding the information and usage of “Developer ID”, please refer to the Remarks section of `S4Control()`.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

**Sample:**

Please refer to the section 4.2.1 “General access”.

## 9.1.3. S4OpenEx

Local: Connect to the specified Senselock ELdevice using specified mode. Connection must be established before any access can be made. This function is the upgrade version of `S4Open()`, for detail, please refer to the Remarks section of this table.
Network:Not supported.

```
DWORD WINAPI S4OpenEx(
  SENSE4_CONTEXT *pS4Ctx,
  S4OPENINFO *pS4OpenInfo);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in,out] Pointer to the `SENSE4_CONTEXT` struct of the Senselock ELdevice to be opened. The content of the struct is returned by `S4Enum()`. |
| *pS4OpenInfo* | [in] Pointer to the `S4OPENINFO` struct, describing the way how the device is to be opened. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

The major difference between this function and `S4Open()` is: `S4Open()` connects the Senselock ELdevice using share mode, namely, after a process has already connected to the Senselock ELdevice, other processes can still connect and make access to that device; `S4OpenEx()`, however, can specify the mode when connection is attempted. When the device is opened using exclusive mode, other process won't be able to connect to the same device concurrently.

`S4OPENINFO` struct is defined as follows:

```
  typedef struct _S4OPENINFO {
    DWORD dwS4OpenInfoSize;
    DWORD dwShareMode;
  } S4OPENINFO;
```

Explanation for each member variable of this struct follows below:

| | |
|---|---|
| *dwS4OpenInfoSize* | Must be `sizeof(S4OPENINFO)`. |
| *dwShareMode* | Device connection mode, two modes are currently supported:<br>• `S4_EXCLUSIZE_MODE` exclusive mode<br>• `S4_SHARE_MODE` share mode |

For more, please refer to the Remarks section of `S4Open()`.
For network verion on operating network, this function will void revoke and return S4_SUCCESS.

**Requirement:**

Hardware version: Senselock EL2.x desktop version, network version（Local operations only）

**Sample:**

```
/* This demonstrates how to open Senselock ELin exclusive mode.*/

  …
  SENSE4_CONTEXT s4ctx = {0};
  S4OPENINFO oinf = {0};
  DWORD ret = 0;

  /* Enumerate device here… */

  /* Open in exclusive mode… */
  oinf.dwS4OpenInfoSize = sizeof(S4OPENINFO );
  oinf.dwShareMode = S4_EXCLUSIZE_MODE;
  ret = S4OpenEx(&s4ctx, &oinf);
  …
```

## 9.1.4. S4Close

Local: Close connection to the device specified.
Network: Close connection to the network version device server.

```
DWORD WINAPI S4Close(SENSE4_CONTEXT *pS4Ctx);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in,out] Pointer to `SENSE4_CONTEXT` struct, pointing to an opened Senselock ELdevice. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

This function closes the connection to the device opened by `S4Open()` or `S4OpenEx()`.

Please notice that this function does NOT clear the security status of Senselock ELdevice, it only closes the connection to that device. If you want to clear the security status of the Senselock ELdevice utterly, please use `S4Control()` to send `S4_RESET_DEVICE` control code to the device before calling this function.

**Requirement:**

Hardware version: Senselock EL2.x desktop version, network version

**Sample:**

Please refer to the section 4.2.1 "General access".

### 9.1.5. S4Control

Local:Sending control code to Senselock ELdevice.
Network:Only support obtaining, releasing authorization, setting overtime.

```
DWORD WINAPI S4Control(
  SENSE4_CONTEXT *pS4Ctx,
  DWORD dwCtlCode,
  VOID *pInBuff,
  DWORD dwInBuffLen,
  VOID *pOutBuff,
  DWORD dwOutBuffLen,
  DWORD *pdwBytesReturned);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to SENSE4_CONTEXT struct, pointing to an opened Senselock ELdevice. |
| *dwCtlCode* | [in] Control code. For currently available values, please refer to the Remarks section. |
| *pInBuff* | [in] Pointer to the input buffer, storing additional data for the control code. Please refer to the Remarks section. |
| *dwInBuffLen* | [in] Length of the data in input buffer *pInBuff*. |
| *pOutBuff* | [out] Pointer to the output buffer, storing the returned information from device after sending the control code. |
| *dwOutBuffLen* | [in] Size of output buffer. |
| *pdwBytesReturned* | [out] Address for a DWORD variable, saving length of the data actually returned to *pOutBuff*, can't be NULL. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Explanation for control codes currently supported by SenseLock EL：
The following control parameter are only supportive in Local API.

- S4_LED_UP
  Purpose：LED up.
  Input：None
  Output：None
- S4_LED_DOWN
  Purpose：LED down.
  Input：None
  Output：None
- S4_LED_WINK
  Purpose：LED wink.
  Input：1 byte integer, standing for the times of LED winking per second.
  Output：None
- S4_GET_DEVICE_TYPE
  Purpose：To get device type.
  Input：None
  Output：1 byte data, indicating device type: desktop version, network version master, or network version slave.
- S4_GET_SERIAL_NUMBER
  Purpose：To get Hardware Unique Serial Number(HUSN).
  Input：None
  Output：8 bytes HUSN.

- S4_GET_VM_TYPE
  Purpose：To get VM type. This control code has been revoked.
- S4_GET_DEVICE_USABLE_SPACE
  Purpose：To get usable space of the device.
  Input：None
  Output：2 bytes WORD integer, standing for the size of usable device space.
- S4_SET_DEVICE_ID
  Purpose：Set ATR information, namely self-defined ID, developer-level authorization is required.
  Input：8 bytes self-defined ID.
  Output：None
- S4_RESET_DEVICE
  Purpose：Reset device thus to clear current security status.
  Input：None
  Output：None
- S4_DF_AVAILABLE_SPACE
  Purpose：To get available space of current directory.
  Input：None
  Output：2 bytes WORD integer, standing for the size of unused space under current directory.
- S4_EF_INFO
  Purpose：To get the attribute information of one file under current directory.
  Input：None
  Output：EFINFO struct, content of which will be illustrated later in this table.
- S4_SET_USB_MODE
  Purpose：Set current device to run on standard USB mode, supported only for Senselock EL2.3 or above .
  Input：None.
  Output：None
- S4_SET_HID_MODE
  Purpose：Set current device to run on standard HID mode, supported only for Senselock EL2.3 or above. This mode makes it unnecessary to install driver under Windows98 or above operating systems, but has lower capability and efficiency than standard USB mode. So it is not recommended.
  Input：None.
  Output：None
- S4_GET_CUSTOMER_NAME
  Purpose：To get “developer ID”(customer ID) of current device, supported only for Senselock EL2.1 or above .
  Input：None.
  Output：4 bytes “developer ID”.
- S4_GET_MANUFACTURE_DATE
  Purpose：To get manufactured date of current device, supported only for Senselock EL2.1 or above.
  Input：None.
  Output：S4_MANUFACTURE_DATE struct, content of which will be illustrated later in this table.
- S4_GET_CURRENT_TIME
  Purpose：To get the current time of the hardware clock, supported only for Senselock EL2.3 or above devices with clock function..
  Input：None.
  Output：a tm struct, which has the same definition as tm struct in C standard running library. For detail, please refer to the definition below or the exemplifications in MSDN.

- S4_SET_NET_CONFIG
  Purpose: After invoking S4CreateDir, S4CreateDirEx to create a netwrok dongle and storing module authorization in the device memory, it is available to use S4Control. By using this parameter, modify the authorization or mode of module in network dongle.
  Input: Point to the handle of S4NETCONFIG contruction.

Output:none.
The following three parameters are only supportive in Network API instead of Local.

- S4_GET_LICENSE
  Purpose: To obtain the authorization of assigned module. Before invoking S4Execute on the client side, S4Control(S4_GET_LICENSE) must be invoked to obtain the authorization. For a module of network dongle's context being opend by S4Open, the authorization can be obtained once successfully. However, the device canbe opened more than once, with multiple context to obtain the authorization.
  Input: To obtain the module ID, a WORD, for instance, to obtain the authorization of module 0, then *pInBuff* imput a handle pointing to WORD 0, *dwInBuffLen* inputs 2.
  Output: None.
- S4_FREE_LICENSE
  Purpose: to release the granted authorization.
  Input:None.
  Output:None.
- S4_MODIFY_TIMOUT
  Purpose:Each client side could assign the time of overdue connection. If it is left blank, then network management tool will take the default value. If during the assigned period, the client side does not have any interaction with server (for execution, obtaining authorizations), then the connection inbetween will be stoped; the authorization obtained will be released; the safe mode will be discharged. The time setting does not affect other contexts.
  Input: The handle pointing to DWORD overtime is measured by seconds.
  Output: None.

EFINFO struct is defined as follows:

```
  typedef struct{
    WORD EfID;
    BYTE EfType;
    WORD EfSize;
  } EFINFO, *PEFINFO;
```

Member variables:

| | |
|---|---|
| *EfID* | File ID. |
| *EfType* | File type |
| *EfSize* | File size. |

tm struct is defined as follows:

```
  struct tm {
    int tm_sec;        /* seconds after the minute - [0,59] */
    int tm_min;        /* minutes after the hour - [0,59] */
    int tm_hour;       /* hours since midnight - [0,23] */
    int tm_mday;       /* day of the month - [1,31] */
    int tm_mon;        /* months since January - [0,11] */
    int tm_year;       /* years since 1900 */
    int tm_wday;       /* days since Sunday - [0,6] */
    int tm_yday;       /* days since January 1 - [0,365] */
    int tm_isdst;      /* daylight savings time flag */
  };
```

S4_MANUFACTURE_DATE struct is defined below:

```
  typedef struct {
    WORD    wYear;
    BYTE    byMonth;
```

```
 BYTE    byDay;
}S4_MANUFACTURE_DATE;
```

Member variables:

| | |
|---|---|
| *wYear* | Year. |
| *byMonth* | Month 1~12. |
| *byDay* | Day, 1~31. |

S4NETCONFIG struct is defined as follows:

```
typedef struct S4NETCONFIG {
DWORD          dwLicenseMode;
DWORD          dwModuleCount;
S4MODULEINFO  ModuleInfo[16];
} S4NETCONFIG;
```

Member variables:

| | |
|---|---|
| *dwLicenseMode* | Assign the mode of module authorization. Two modes: Process mode, in obtaining any modules authorization, it will take anthorization of this moduel; IP mode, then any authorizations obtained by same module on the same machine only take as one authorization by sharing. |
| *dwModuleCount* | Assigned module quantity. Max. 16. |
| *ModuleInfo* | Info in detail of assigned module. Please refer to S4MODULEINFO structural definition. |

S4MODULEINFO struct is defined as follows:

```
typedef struct S4MODULEINFO {
 WORD         wModuleID;
 WORD         wLicenseCount;
} S4MODULEINFO;
```

Member variables:

| | |
|---|---|
| *wModuleID* | Module's ID (0-65535) |
| *wLicenseCount* | Total amount of authorization of assigned module ( 0-65535) |

**Requirement:**

Hardware version:Senselock EL2.x desktop version, network version（Obtaining authorization, releasing authorization, setting connection, other operations are locally operatable）.

**Sample:**

Please refer to the section 4.2.1 "General access".

## 9.1.6. S4CreateDir

Local: Create a new directory. If creation succeeds, the newly created directory will be selected to be "current directory". Using `S4CreateDirEx()` instead of this function is recommended.
Network: Not supported

```
DWORD WINAPI S4CreateDir(
  SENSE4_CONTEXT *pS4Ctx,
  LPCSTR lpszDirID,
  DWORD dwDirSize,
  DWORD dwFlags);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to `SENSE4_CONTEXT` struct, pointing to an opened Senselock ELdevice. |
| *lpszDirID* | [in] Directory ID. In Senselock ELAPI, ID of file or directory is passed in using a string. For example, string "1234"means that the ID of directory or file is 0x1234. Please notice that the inputted ID must contain 4 chars and be in hexadecimal format. For example, to the directory ID 0x0012, you must input "0012" instead of "12". For network version, the parameter can be only set as "\", to create root directory only but others. For system reserved IDs, please refer to the Remarks section of this table. |
| *dwDirSize* | [in] Size of the directory to be created, can't exceed the usable space of current directory. When creating directory, the file system itself will take certain amount of space: for desktop version, it will take 197 bytes. For example, if one specifies 1024 bytes as the total space when creating the directory, the actual usable space after creation will be: desktop version `(1024-197)=827.` Additionally, directory's file header will also take certain amount of its parent directory's space(42 bytes).For example, if one creates a sub directory of 1024 bytes under root directory, it actually takes `1024+42=1066` bytes space of the root directory totally. |
| *dwFlags* | [in] Flag options for creating a directory.<br>• `S4_CREATE_ROOT_DIR`<br>Create root directory. When this flag is set, directory ID must be `"\"`,*dwDirSize* must be `0,`or there will be parameter error.<br>• `S4_CREATE_SUB_DIR`<br>Create sub directory, effective only for desktop version.<br>• `S4_CREATE_MODULE`<br>Create a sub module, effective only for network version. Sub module of network version is like a sub directory, it can ONLY be created directly under the root directory<br>After the sub module is created, its license number will be set to be 10 automatically. You can use `S4WriteFile()` to modify the license number. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

This function does NOT create ATR file automatically. For details regarding ATR file, please refer to the Remarks section of `S4CreateDirEx()`.

After a directory is created, its developer PIN and user PIN will be set to be default value, namely, developer PIN: `"123456781234567812345678"`, user PIN: `"12345678"`, you MUST modify the developer PIN to be a secret value. For details about PINs ,please refer to section 1.3.1

After using this function to create root directory in network version, the total authorization of the device is 10, authorization mode will be Process mode, 3 modules by default, 5 authorizations for each module (ID: 0, 1, 2)

For network version, root directory is only creatabe but any other subdirectories.
For standalone version, max 3 level file structure can be created (root directory being taken as level one, like \0001\0002).

Some IDs are reserved by system, these include：`0x0000,0x3f00 (reserved as ID of root directory).`

The ID of any directory's parent directory, sibling directory and subdirectory can not be same.
For network version, the function is void invoked, and returen S4_SUCCESS.

**Requirement:**

Hardware version: Senselock EL2.x desktop version, network version（Local operations only）

**Sample:**

Refer to the section 4.2.2 "Developer level access"

## 9.1.7. S4CreateDirEx

Local: Create a new directory. If creation succeeds, the newly created directory will be selected to be "current directory".
Network: Not supported

```
DWORD WINAPI S4CreateDirEx(
  SENSE4_CONTEXT *pS4Ctx,
  LPCSTR lpszDirID,
  DWORD dwDirSize,
  DWORD dwFlags,
  S4CREATEDIRINFO *pCreateDirInfo);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to `SENSE4_CONTEXT` struct, pointing to an opened Senselock ELdevice. |
| *lpszDirID* | [in] Directory ID. In Senselock ELAPI, ID of file or directory is passed in using a string. For example, string "1234"means that the ID of directory or file is 0x1234. Please notice that inputted ID must contain 4 chars and be in hexadecimal format. For example, to the directory ID 0x0012, you must input "0012" instead of "12". For network version, the parameter can be only set as "\", to create root directory only but others. For system reserved IDs, please refer to the Remarks section of this table. |
| *dwDirSize* | [in] Size of the directory to be created, can't exceed the remaining space of current directory. When creating directory, the file system itself will take certain amount of space: for desktop version, it will take 197 bytes. For example, if one specifies 1024 bytes as the total space when creating the directory, the actual usable space after creation will be: desktop version `(1024-197)=827.` Additionally, directory's file header will also take certain amount of its parent directory's space(42 bytes).For example, if one creates a sub directory of 1024 bytes under root directory, it actually takes `1024+42=1066` bytes space of the root directory totally. |
| *dwFlags* | [in] Flag options for creating a directory.<br>• `S4_CREATE_ROOT_DIR`<br>Create root directory. When this flag is set, directory ID must be `"\"`,*dwDirSize* must be `0`,or there will be parameter error.<br>• `S4_CREATE_SUB_DIR`<br>Create sub directory, effective only for desktop version.<br>• `S4_CREATE_MODULE`<br>Create a sub module, effective only for network version. Sub module of network version is like a sub directory, it can ONLY be created directly under the root directory<br>After the sub module is created, its license number will be set to be 10 automatically. You can use `S4WriteFile()` to modify the license number. |
| *pCreateDirInfo* | [in] This parameter is a pointer to the `S4CREATEDIRINFO` struct. You must fill in this struct if *dwFlags* is set to be `S4_CREATE_ROOT_DIR` . |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Using this API to create root directory will automatically create an ATR file which is used to store self-defined ID (the content of this ID is from the member variable *szAtr[8]* of struct *pCreateDirInfo*).
ATR file is of special type, and its content has only 8 bytes. If this file exists, then when using `S4Enum()` to list devices, *bID* member variable of `SENSE4_CONTEXT` struct will be the 8 octets content of ATR file. If you want the real 8 octets HUSN, you must use `S4Control()` to send a `S4_GET_SERIAL_NUMBER` control code. Using ATR file can well distinguish devices from different developers. For detail, please refer to the Remarks section of `S4Enum()`.

After a directory is created, its developer PIN and user PIN will be set to be default value, namely, developer PIN: `"123456781234567812345678"`, user PIN: `"12345678"`, you MUST modify the developer PIN to be a secret value. For details about PINs ,please refer to section 1.3.1

Some IDs are reserved by system, these include：`0x0000,0x3f00 (reserved as ID of root directory).`

The ID of any directory’s parent directory, sibling directory and subdirectory can not be same.
For network version, the function is void invoked, and returen S4_SUCCESS.

`S4CREATEDIRINFO` struct is defined as follows：

```
typedef struct _S4CREATEDIRINFO {
  DWORD dwS4CreateDirInfoSize;
  BYTE  szAtr[8];
  S4NETCONFIG NetConfig;
} S4CREATEDIRINFO;
```

Explanation for each member variable of this struct follows below:

| | |
|---|---|
| *dwS4CreateDirInfoSize* | Size of the struct, must be: `sizeof(S4CREATEDIRINFO).` |
| *szAtr* | 8 octets content of ATR file, namely self-defined ID. |
| *NetConfig* | S4NETCONFIG struct, please refer to the note of S4Control about S4NETCONFIG. It should be regardless to network version. |

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version（Local operations only）

**Sample:**

```
/* This demonstrates how to create root dir with developer
  specific identifier. */

 …
```

```
SENSE4_CONTEXT s4ctx = {0};
S4CREATEDIRINFO rinf = {0};
DWORD ret = 0;

/* Open and verify developer PIN here. Delete original root
  dir if exists… */

/* Create new root dir… */
rinf.dwS4CreateDirInfoSize = sizeof(S4CREATEDIRINFO);
memcpy(rinf.szAtr, “SenseLock EL”, 8);
ret = S4CreateDirEx(&s4ctx, “\\”, 0,
          S4_CREATE_ROOT_DIR,&rinf);
…
```

### 9.1.8. S4ChangeDir

Local: Select a new working directory.
Network: Supports invoking voidly.

```
DWORD WINAPI S4ChangeDir(
  SENSE4_CONTEXT *pS4Ctx,
  LPCSTR lpszPath);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to SENSE4_CONTEXT struct,pointing to an opened Senselock ELdevice. |
| *lpszPath* | [in] New working directory path. There are two ways for path selecting:<br>• Absolute path, based on the root directory, for example "\0001\0012"、"\"、"\0011" etc;<br>• Relative path, based on current directory, for example "0001\0012"、"00e4".<br>For local operations of network version, as only root directory available, this function is only way to switch the root directory. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

When change to another directory of upper level or the same level, the security status of the original directory will be lost. Re-authentication is required if you want to re-select the original directory.

The ID of any directory's parent directory, sibling directory and subdirectory can not be same.
For network version, the function is void invoked, and returen S4_SUCCESS.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

**Sample:**

Please refer to the section 4.2.1 "General access".

## 9.1.9. S4EraseDir

Local:Erase all the content of current directory, and reset it's security attributes.
Network:Not supported

```
DWORD WINAPI S4EraseDir(
  SENSE4_CONTEXT *pS4Ctx,
  LPCSTR lpszPath);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to SENSE4_CONTEXT struct,pointing to an opened Senselock ELdevice. |
| *lpszPath* | [in] Reserved, must be NULL. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

If current directory is a sub directory, this function will erase all the files and children directories of current directory(current directory itself will NOT be deleted). The developer PIN and user PIN of current directory will also be restored to default values.

If current directory is a root directory, this function will delete the root directory directly, and that will make the dongle to get back to empty status.

For network version, it is only workable to delete whole root directory.

The ID of any directory's parent directory, sibling directory and subdirectory can not be same.
For network version, the function is void invoked, and returen S4_SUCCESS.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version（Local operations only）

**Sample:**

Refer to the section 4.2.2 "Developer level access"

## 9.1.10.S4VerifyPin

Local: Verify PINs(developer level or user level) of current directory so as to get corresponding access privileges.
Network:Supports invoking voidly

```
DWORD WINAPI S4VerifyPin (
  SENSE4_CONTEXT *pS4Ctx,
  BYTE *pbPin,
  DWORD dwPinLen,
  DWORD dwPinType);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to `SENSE4_CONTEXT` struct,pointing to an opened Senselock ELdevice. |
| *pbPin* | [in] Pointer to the PIN array. |
| *dwPinLen* | [in] Length of the PIN, developer PIN is 24 octets, and user PIN is 8 octets. |
| *dwPinType* | [in] PIN type to be verified: |

- `S4_DEV_PIN` Verify developer PIN
- `S4_USER_PIN` Verify user PIN

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Senselock ELmay be locked after several times of failed attempts in verifying developer level PIN. When the hardware is locked, even Senselock can’t unlock it.

To avoid accidentally being locked, the developer level PIN of Senselock ELtakes measures of “blocking strategy”: If there are errors in the former 16 bytes of the developer PIN, it won’t result in being locked. If the former 16 bytes of the developer PIN is correct, and there are errors in the latter 8 bytes, several attempts will make Senselock ELto be locked. In the latter case, function’s return value will be: `0x63CX`, in which `X` is between `0~F` (means the remaining times of retrials allowed before the Senselock EL‘s being locked) For example, if the function returns `0x63CA`, it means that 10 attempts can be made before the dongle’s being locked.

No matter how many failed attempts have been made to verify user PIN, it NEVER locks the dongle.
All the PINs are effective only for current directory, for details, please refer to the section 1.3.1 “PIN”

Local API only authenticate the PIN to root directory on network version.

For network version, the function is void invoked, and returen S4_SUCCESS.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version（Local operations only）

**Sample:**

Refer to the section 4.2.2 “Developer level access” and section 4.2.3 “User level access”

### 9.1.11.S4ChangePin

Local: Change PINs(developer level or user level) of current directory .
Network:Not supported

```
DWORD WINAPI S4ChangePin (
  SENSE4_CONTEXT *pS4Ctx,
  BYTE *pbOldPin,
  DWORD dwOldPinLen,
  BYTE *pbNewPin,
  DWORD dwNewPinLen,
  DWORD dwPinType);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to SENSE4_CONTEXT struct,pointing to an opened Senselock ELdevice. |
| *pbOldPin* | [in] Pointer to the old PIN |
| *dwOldPinLen* | [in] Length of the old PIN, developer PIN is 24 octets, and user PIN is 8 octets; |
| *pbNewPin* | [in] Pointer to the new PIN. |
| *dwNewPinLen* | [in] Length of the new PIN, developer PIN is 24 octets, and user PIN is 8 octets; |
| *dwPinType* | [in] Type of the PIN to be modified: |

| | |
|---|---|
| • S4_DEV_PIN | change developer level PIN |
| • S4_USER_PIN | change user level PIN |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

This function will automatically verify the old PIN so as to get corresponding security authorization, so all the rules regarding PIN locking in the Remarks section of S4VerifyPin() apply to this function too.

After PIN changing, corresponding security authorization will be obtained automatically; PIN changing can be done only at the same level, namely, developer level PIN can’t be used to modify user level PIN.

Developer level authorization of current directory(network version module) must be acquired before changing authentication PIN of network version. For details about authentication PIN of network version, please refer to “Direction for network version usage”.

All the PINs are effective only for current directory, for details, please refer to the section 1.3.1 “PIN”

Local API only authenticate the PIN to root directory on network version.

For network version, the function is void invoked, and returen S4_SUCCESS.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version （Local operations only）

**Sample:**

Refer to the section 4.2.2 “Developer level access”

## 9.1.12.S4WriteFile

Local:Create, update file or modify license number etc.under current directory.
Network:Not supported

```
DWORD WINAPI S4WriteFile (
  SENSE4_CONTEXT *pS4Ctx,
  LPCSTR lpszFileID,
  DWORD dwOffset,
  LPVOID lpBuffer,
  DWORD dwBufferSize,
  DWORD dwFileSize,
  DWORD *pdwBytesWritten,
  DWORD dwFlags,
  BYTE bFileType);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to SENSE4_CONTEXT struct,pointing to an opened Senselock ELdevice. |
| *lpszFileID* | [in] File ID,there are special stipulation when *dwFlags* is any of the following values:<br>• S4_KEY_GEN_RSA_FILE<br>Need to input IDs of two files, 8 bytes in total, the former 4 bytes is the ID of a public file and the latter 4 bytes is the ID of corresponding private file.<br>• S4_SET_LICENCES<br>Must be NULL. |
| *dwOffset* | [in] Offset for file writing, when *dwFlags* is any of the following values, it must be 0:<br>• S4_KEY_GEN_RSA_FILE<br>• S4_SET_LICENCES<br>• S4_RSA_PUBLIC_FILE<br>• S4_RSA_PRIVATE_FILE |
| *lpBuffer* | [in] Data content to be written in. Data format has relation with file type. Please refer to explanation for the file types in the Remarks section of this table.<br>When *dwFlags* is any of the following values, it must be NULL;<br>• S4_KEY_GEN_RSA_FILE<br>• S4_SET_LICENCES<br>When *dwFlags* is any of the following values, you need to input keys with required format. For details, please refer to the Remarks section of this table:<br>• S4_RSA_PUBLIC_FILE<br>• S4_RSA_PRIVATE_KEY |
| *dwBufferSize* | [in] Length of the data to be written into the file. There are special stipulation when *dwFlags* is any of the following values:<br>• S4_KEY_GEN_RSA_FILE<br>dwBufferSize = 0<br>• S4_SET_LICENCES<br>dwBufferSize = 0<br>• S4_RSA_PUBLIC_FILE<br>dwBufferSize = |

| | |
|---|---|
| | `sizeof(S4_RSA_PUBLIC_KEY)`<br>• `S4_RSA_PRIVATE_FILE`<br>`dwBufferSize =`<br>`sizeof(S4_RSA_PRIVATE_KEY)` |
| *`dwFileSize`* | [in] Size of the file to be created or license number, depending on the value of *`dwFlags`*:<br>• `S4_CREATE_NEW`<br>`dwFileSize` is the size of the file to be created. Please notice that, to public file, it must be 136; to private file, it must be 330.<br>• `S4_SET_LICENCES`<br>*`dwFileSize`* is the license number to be set, must be between 1 and 255.<br>• `Else`<br>Must be `0`. |
| *`pdwByteWritten`* | [out] Length of data actually written(in byte) |
| *`dwFlags`* | [in] Flag options, please see Remarks section for details. |
| *`bFileType`* | [in] File type,please see Remarks section for details. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Available values for flag option（*`dwFlags`*）are listed below:

| | |
|---|---|
| `S4_CREATE_NEW` | Create new file. By default, attribute of the newly created executable file will be set to “internally read/writable”. |
| `S4_FILE_EXECUTE_ONLY` | Combined with flag `S4_CREATE_NEW` to create an “internally unreadable/unwritable” executable file, this flag is effective only for executable files. |
| `S4_CREATE_PEDDING_FILE` | For Senselock ELhardware version 2.3 or higher, this flag is ignored. |
| `S4_UPDATE_FILE` | Update an existing file. |
| `S4_KEY_GEN_RSA_FILE` | Generate a RSA key pair, can be used together with `S4_CREATE_NEW` |
| `S4_SET_LICENCES` | Set license number for network version dongles. |

* “Internally readable/writable” and “internally unreadable/unwritable” mean whether or not the objective file can be operated by SES `_read()` and `_write()`. For detail, please refer to “the first chapter section 1.3.2”.

Available values for file type（*`bFileType`*）are listed below:

| | |
|---|---|
| `S4_EXE_FILE` | VM executable, binary objective file generated by Keil C51 or Skit compiler.<br>Data process: Write the data in *`lpbuffer`* into the objective file using binary format. |
| `S4_XA_EXE_FILE` | XA executable, binary objective file generated by Rkit.<br>Data process: Write the data in *`lpbuffer`* into the objective file using binary format. |

| | |
|---|---|
| S4_DATA_FILE | Data file<br>Data process: Write the data in *lpbuffer* into the objective file using binary format. |
| S4_RSA_PUBLIC_FILE | RSA public key file*.<br>Data process: Write the data in *lpbuffer* into the objective file after conversion, see the comments below this table |
| S4_RSA_PRIVATE_FILE | RSA private file.<br>Data process: Write the data in *lpbuffer* into the objective file after conversion, see the comments below this table. |
| 0 | If *dwFlags* is S4_KEY_GEN_RSA_FILE or S4_SET_LICENCES,*bFileType* must be 0 |

* When writing public or private file, API does NOT write the inputted data to the file using binary format directly, instead, it converts the inputted data content first. The data inputted must be of public key S4_RSA_PUBLIC_KEY format and of private key S4_RSA_PRIVATE_KEY format as defined in Senselock ELAPI's header file; To files of other types, the function write directly the inputted data using binary format.

A file can't be erased directly once it has been created. If you want to delete a file and free the storage space it takes completely, you must delete or empty the directory under which the file resides.

Some IDs are reserved by system, these include：0x0000, 0x0015, 0x0016, 0x0018,0x001e,0x3f00,0x3f01,0x3f02,0x3f03,0x3f04.

For network version, the function is void invoked, and returen S4_SUCCESS.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version（Local operations only）

**Sample:**

```
/* This demonstrates how to create a 1K bytes data file
  and write some information into it. */

  ……
  SENSE4_CONTEXT s4ctx = {0};
  unsigned char buff[] = "This is a test message!";
  DWORD dwRet = 0;
  DWORD dwBytesWritten = 0;

  /* Open and verify developer PIN here… */

  /* Now create a new data file and write it. */
  dwRet = S4WriteFile(
      &s4ctx,
      "d001",    /* New file ID. */
      0,       /* Write offset. */
      buff,      /* Data to write. */
      sizeof(buff),/* Number of bytes to write. */
      1024,      /* New file size, 1K bytes here. */
      &dwBytesWritten,
      S4_CREATE_NEW,/* Specify to create a new file. */
```

```
        S4_DATA_FILE);/* File type. */

    if (dwRet != S4_SUCCESS)
    {
      /* Handle error here… */
    }
    ……
```

## 9.1.13.PS4WriteFile

Local: Create or update a file under current directory. This API encapsulate `S4WriteFile()` and operate directly the disk file which simplifies greatly the process of initializing Senselock ELfile system. All the work done by this API can also be achieved by `S4WriteFile()`.
This function is not a fundamental Senselock ELAPI, so only C version and Win32 dynamic link library of it are provided. It's declared in the file *psense4.h*.
Network:Not supported

```
DWORD WINAPI PS4WriteFile (
  SENSE4_CONTEXT *pS4Ctx,
  LPCSTR lpszFileID,
  LPCSTR lpszPCFilePath,
  DWORD *dwFileSize,
  DWORD dwFlags,
  DWORD dwFileType,
  DWORD *pdwBytesWritten);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to `SENSE4_CONTEXT` struct,pointing to an opened Senselock ELdevice. |
| *lpszFileID* | [in] File ID. |
| *lpszPCFilePath* | [in] Disk file path, and the content of the file is the data to be written into the dongle. There are different requirements for the format of disk file according to different objective file types, please refer to the explanation for file types in Remarks section of this table. If you want just to create a new file without writing in any content, you can set this parameter to be `NULL`. |
| *dwFileSize* | [in,out] Size of the file to be created. If you want to determine automatically the size according to the type and content of disk file, you can set this parameter to be 0; If you want to update an existing file, this parameter MUST be set to 0. |
| *dwFlags* | [in] Flag options, please see Remarks section for details. |
| *bFileType* | [in] File type, please see Remarks section for details. |
| *pdwByteWritten* | [out] Length of data actually written(in byte) |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Available values for flag option（*dwFlags*）are listed below:

| | |
|---|---|
| S4_CREATE_NEW | Create new file. By default, attribute of the newly created executable file will be set to "internally read/writable". |
| S4_FILE_EXECUTE_ONLY | Combined with flag `S4_CREATE_NEW` to create an "internally unreadable/unwritable" executable file, this flag is effective only to executable files. |
| S4_CREATE_PEDDING_FILE | For Senselock ELhardware version 2.3 or higher, this flag is ignored. |

| | |
|---|---|
| S4_UPDATE_FILE | Update an existing file. |

* "Internally readable/writable" and "internally unreadable/unwritable" means whether or not that file can be operated by SES _read() and _write(). For detail, please refer to the first chapter section 1.3.2.

Available values for file type（*bFileType*）are listed below:

| | |
|---|---|
| S4_EXE_FILE | VM executable, binary objective file generated by Keil C51 or Skit compiler.<br>Data process: Write the data in disk file into the objective file using binary format. |
| S4_XA_EXE_FILE | XA executable, binary objective file generated by Rkit.<br>Data process: Write the data in disk file into the objective file using binary format. |
| S4_HEX_FILE | VM executable, binary objective file generated by Keil C51 or Skit compiler. The difference between this value and S4_EXE_FILE is that when this file type is specified, the disk file inputted will be of HEX format and API will internally convert it to binary format automatically.<br>Data process: Convert the data in disk file and then write them into the objective file. |
| S4_XA_HEX_FILE | XA executable, binary objective file generated by Rkit. The difference between this value and S4_XA_EXE_FILE is: that when this file type is specified, the disk file inputted will be of HEX format and API will internally convert it to binary format automatically.<br>Data process: Convert the data in disk file and then write them into the objective file. |
| S4_DATA_FILE | Data file<br>Data process: Write the data in disk file into the objective file using binary format. |
| S4_RSA_PUBLIC_FILE | RSA public key file, the data contained in the disk file must be of public key S4_RSA_PUBLIC_KEY format as defined in Senselock ELAPI's header file;<br>Data process: Convert the data in disk file and then write them into the objective file. |
| S4_RSA_PRIVATE_FILE | RSA private file, the data contained in the disk file must be of private key S4_RSA_PRIVATE_KEY format as defined in Senselock ELAPI's header file;<br>Data process: Convert the data in disk file and then write them into the objective file. |

A file can't be erased directly once it has been created. If you want to delete a file and free the storage space it takes completely, you must delete or empty the directory under which the file resides.

Some IDs are reserved by system, these include：0x0000, 0x0015, 0x0016, 0x0018,0x001e,0x3f00,0x3f01,0x3f02,0x3f03,0x3f04.

For network version, the function is void invoked, and returen S4_SUCCESS.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version（Local operations only）

**Sample:**

```
/* This demonstrates how to create a new XA EXF file
  and write the binary code into it.
  Suppose that the XA EXF code is stored as
  "c:\demo\demo1.hex", which is in Intel hex format. */

  ……
  SENSE4_CONTEXT s4ctx = {0};
  DWORD dwRet = 0;
  DWORD dwBytesWritten = 0;
  DWORD dwFileSize = 0;


  /* Open and verify developer PIN here… */

  /* Now create a new XA EXF file and write it. */
  dwRet = PS4WriteFile(
      &s4ctx,
      "E001",
      "c:\\demo\\demo1.hex",/* Disk file. */
      &dwFileSize,  /* 0, auto detect file size. */
      S4_CREATE_NEW | S4_FILE_EXECUTE_ONLY,/* No R/W. */
      S4_XA_HEX_FILE,   /* File type. */
      &dwBytesWritten);

  if (dwRet != S4_SUCCESS)
  {
    /* Handle error here… */
  }
  ……
```

### 9.1.14.S4Execute

Local:Execute VM EXF specified under current directory.
Network:Execute VM EXF specified under current module of remote network dongle.

```
DWORD WINAPI S4Execute (
  SENSE4_CONTEXT *pS4Ctx,
  LPSCTR lpszFileID,
  LPVOID lpInBuffer,
  DWORD dwInBufferSize,
  LPVOID lpOutBuffer,
  DWORD dwOutBufferSize,
  DWORD *dwBytesReturned);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to `SENSE4_CONTEXT` struct,pointing to an opened Senselock ELdevice. |
| *lpszFileID* | [in] ID of the EXF to be executed. |
| *lpInBuffer* | [in] Input buffer, storing the data to be transmitted in, namely the content of communication buffer when input |
| *dwInBufferSize* | [in] Length of the inputted data |
| *lpOutBuffer* | [out] Output buffer, storing the data to be transmitted out, namely the content of communication buffer when output |
| *dwOutBufferSize* | [in] Size of *lpOutBuffer*, when it is less than the length of the data returned from inside SenseLock EL, this function will return error indicating insufficient output buffer space. |
| *dwBytesReturned* | [out] Length of the data actually returned from SenseLock EL, namely the length of the data in communication buffer. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

You must verify user PIN before invoking this API.

This API can ONLY execute VM EXF, if you want to execute XA EXF, please use `S4ExecuteEx()` instead.

For local operations, the input buffer is max 250 bytes; for network operations, input buffer is max 237 bytes.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

**Sample:**

Refer to the section 4.2.2 “User level access”

## 9.1.15.S4ExecuteEx

Local:Execute VM EXF or XA EXF specified under current directory.
Network:Execute Not supported.

```
DWORD WINAPI S4ExecuteEx (
  SENSE4_CONTEXT *pS4Ctx,
  LPSCTR lpszFileID,
  DWORD dwFlag,
  LPVOID lpInBuffer,
  DWORD dwInBufferSize,
  LPVOID lpOutBuffer,
  DWORD dwOutBufferSize,
  DWORD *dwBytesReturned);
```

**Parameters:**

| | |
|---|---|
| *pS4Ctx* | [in] Pointer to `SENSE4_CONTEXT` struct,pointing to an opened Senselock ELdevice. |
| *lpszFileID* | [in] ID of the EXF to be executed. |
| *dwFlag* | [in] Type of the executable: • `S4_VM_EXE` VM executable file • `S4_XA_EXE` XA executable file |
| *lpInBuffer* | [in] Input buffer, storing the data to be transmitted in, namely the content of communication buffer when input |
| *dwInBufferSize* | [in] Length of the inputted data |
| *lpOutBuffer* | [out] Output buffer, storing the data to be transmitted out, namely the content of communication buffer when output |
| *dwOutBufferSize* | [in] Size of *lpOutBuffer*, when it is less than the length of the data returned from inside SenseLock EL, this function will return error indicating insufficient output buffer space. |
| *dwBytesReturned* | [out] Length of the data returned from SenseLock EL, namely the length of the data in communication buffer. |

**Return values:**

Return S4_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

You must verify user PIN before invoking this API.

Except for the support to XA EXF, this API is almost identical with `S4Execute()`.

While executing VM EXF file, input buffer is max 250 bytes. While executing XA EXF file, input buffer is max 248 bytes.

For network version, the function is void invoked, and returen S4_SUCCESS.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

**Sample:**

```
/* Considering the sample code of section 4.2.3, we can add a flag(in
RED) in the code to indicate the destination file is a XA EXF. */
```

```
/* Invoke exf 0xd001. */
ret = S4ExecuteEx(&s4ctx, “d001”, S4_XA_EXE
        input, input_len, output, output_len, &len);

if (ret != S4_SUCCESS)
{
  printf("Invoke 0xd001 failed! <error code = 0x%08x>\n", ret);
  S4Close(&s4ctx);
  return 1;
}
```

### 9.1.16.S4Startup

### 9.1.17.S4Cleanup

These functions has already been revoked, please do not use.

## 9.2. Error Code Index

| Error code | Value | Comment |
|---|---|---|
| S4_SUCCESS | 0x00000000 | Operation successful. |
| S4_UNPOWERED | 0x00000001 | The device has been powered off |
| S4_INVALID_PARAMETER | 0x00000002 | Invalid parameter. |
| S4_COMM_ERROR | 0x00000003 | Communication error, i.e.data transmission timeout. |
| S4_PROTOCOL_ERROR | 0x00000004 | Wrong communication protocol. |
| S4_DEVICE_BUSY | 0x00000005 | Device busy. |
| S4_KEY_REMOVED | 0x00000006 | Device removed or not connected. |
| S4_INSUFFICIENT_BUFFER | 0x00000011 | Insufficient buffer. |
| S4_NO_LIST | 0x00000012 | No device is found. |
| S4_GENERAL_ERROR | 0x00000013 | Commonly indicates not enough memory |
| S4_UNSUPPORTED | 0x00000014 | Function not supported or file system has not been created. |
| S4_DEVICE_TYPE_MISMATCH | 0x00000020 | Device type mismatch. |
| S4_FILE_TYPE_MISMATCH | 0x00006981 | File type mismatch. |
| S4_FILE_SIZE_CROSS_7FFF | 0x00000021 | (Applied only to version 2.2 or earlier). |
| S4_CURRENT_DF_ISNOT_MF | 0x00000201 | The net module to be created isn't child directory of the root directory, available for network version only. |
| S4_INVAILABLE_MODULE_DF | 0x00000202 | The current directory is not a module, available only for network version. |
| S4_FILE_SIZE_TOO_LARGE | 0x00000203 | The file size is larger than 0x7FFF. |
| S4_DF_SIZE | 0x00000204 | The size of the specified directory is not enough. |
| S4_DIRECTORY_EXIST | 0x00006901 | Directory already exists; max level of directory is exceeded; the directory is built completely; Repetitive creating root directory; unset root directory |
| S4_FILE_TYPE_MISMATCH | 0x00006981 | The device type does not match |
| S4_INSUFFICIENT_SECU_STATE | 0x00006982 | Security status not satisfied, corresponding PIN must be verified. |
| S4_PIN_BLOCK | 0x00006983 | Device locked. |
| S4_APPLICATION_TEMP_BLOCK | 0x00006985 | Application temporarily locked. |
| S4_FILE_EXIST | 0x00006a80 | File already exists. |
| S4_DEVICE_UNSUPPORTED | 0x00006a81 | Unsupported by the device. |
| S4_FILE_NOT_FOUND | 0x00006a82 | File not found. |
| S4_INSUFFICIENT_SPACE | 0x00006a84 | Insufficient file space. |
| S4_OFFSET_BEYOND | 0x00006B00 | File offset exceeds the boundary. |
| S4_CRYPTO_KEY_NOT_FOUND | 0x00009403 | Cryptographic key not found. |
| S4_APPLICATION_PERM_BLOCK | 0x00009303 | The directory has been locked. |
| S4_DATA_BUFFER_LENGTH_ERROR | 0x00006700 | Invalid data length |
| S4_CODE_RANGE | 0x00010000 | Out of code range, generally result from stack overflow. |
| S4_CODE_RESERVED_INST | 0x00020000 | Illegal instruction. |
| S4_CODE_RAM_RANGE | 0x00040000 | Illegal pointer to internal RAM. |
| S4_CODE_BIT_RANGE | 0x00080000 | Illegal bit variable. |
| S4_CODE_SFR_RANGE | 0x00100000 | Illegal SFR. |
| S4_CODE_XRAM_RANGE | 0x00200000 | Illegal pointer to external RAM. |
| | | |
| S4WF_INVALID_S4CONTEXT | 0x00000101 | Invalid S4Context parameter(PS4WriteFile). |
| S4WF_INVALID_FILE_ID | 0x00000102 | Invalid File ID (PS4WriteFile). |
| S4WF_INVALID_PC_FILE | 0x00000103 | Invalid PC file(PS4WriteFile). |
| S4WF_INVALID_FLAGS | 0x00000104 | Invalid dwFlags parameter(PS4WriteFile). |
| S4WF_INVALID_FILE_SIZE | 0x00000105 | Invalid dwFileSize parameter (PS4WriteFile). |
| S4WF_INVALID_FILE_TYPE | 0x00000106 | Invalid dwFileType parameter (PS4WriteFile). |
| | | |
| S4_MODULE_NOT_FOUND | 0x00000301 | Failed to find assigned module |
| S4_LICENSE_EXIST | 0x00000302 | License granted already |
| S4_USER_NOT_FOUND | 0x00000303 | No users found in network version |
| S4_LICENSE_INVALID | 0x00000304 | Non-effective license |
| S4_TIMEOUT | 0x00000305 | Overdue operating on network version |

| | | |
|---|---|---|
| S4_NETWORK_ERROR | 0x00000306 | Network error |
| S4_LICENSE_NOT_FOUND | 0x00000307 | No available license |
| S4_EXECUTE_ERROR | 0x00000308 | Execution error |
| S4_TOTALLICENSE_BEYOND | 0x00000309 | Exceeding total licenses |
| S4_MODULELICENSE_BEYOND | 0x00000310 | Exceeding total licenses of module |
| S4_DEVICE_INVALID | 0x00000311 | Device is not available |
| S4_USERPIN_ERROR | 0x00000312 | Verification error on User PIN |
| S4_MODULE_ZERO | 0x00000313 | Module number is 0 |
| S4_DEVICETYPE_ERROR | 0x00000314 | Device type error |
| S4_DEVICE_START_FAILED | 0x00000315 | Device initiating error |
| S4_DEVICE_STOP_FAILED | 0x00000316 | Failed to stop device |
| S4_NET_TIMEOUT | 0x00000324 | Overdue communication on network version |
| | | |
| S4_ERROR_UNKNOWN | 0xffffffff | Unknown error. |

# Appendix A. Cryptographic Algorithms

Senselock ELhardware alone can provide DES/TDES、RSA、SHA1 algorithms. In real applications however, the calculation procedure for some cryptographic algorithms must be done on PC end. For example, you may hope to encrypt part of your file to ciphertext before software release, and decrypt it using Senselock ELwhen running this software. In this case, you can use software algorithms to complete the encryption process.

Algorithms in Senselock ELhardware are all of standard ones, so you can easily get various implementations of them. That’s one of the key features a secure algorithm must possess: it must not depend on the algorithm itself, instead, it must depend on the confidentiality of the secret key. That’s also what distinguishes Senselock ELfrom other common dongles which often provide one so-called “black box algorithm” and security of it by no means can be evaluated.

SDK provides the software implementations of all the algorithms mentioned above to make it convenient for corresponding development. You can use these algorithms’ interface directly or use other implementations obtained by yourself. They will do the same work.

Besides, to simplify the generation ,conversion and other operations of secret keys, SDK also provides some key-operating function interfaces.

Corresponding libraries and header files are under the product’s installation directory *%SDK%\support\cryptolib*. Only C-version API is offered currently.

## A.1. Key Operating Functions

### A.1.1. X_GenerateRsaKeys

Generate a RSA key pair on PC and convert it to the public/private key format supported by Senselock ELhardware.

```
int WINAPI X_GenerateRsaKeys(
  COS_RSA_PUBLIC_KEY *pCosPubKey,
  COS_RSA_PRIVATE_KEY *pCosPriKey);
```

**Parameters:**

| | |
|---|---|
| *pCosPubKey* | [out] Pointer to `COS_RSA_PUBLIC_KEY` struct which defines the format of RSA public key inside SenseLock EL. For detail, please refer to the Remarks section of "`rsa_enc()`"in the 8th chapter. |
| *pCosPriKey* | [out] Pointer to `COS_RSA_PRIVATE_KEY` struct which defines the format of RSA private key inside SenseLock EL. For detail, please refer to the Remarks section of "`rsa_enc()`"in the eigthth chapter. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

If you want to perform RSA calculation internally in Senselock ELhardware, a RSA key pair of corresponding format must be provided. You can use API `S4_WriteFile()` to generate a key pair inside the hardware or use this API(`X_GenerateRsaKeys()`) to generate a key pair and then import them using binary mode into public/private key files inside the hardware. Please notice that the RSA private key generated inside the hardware can NEVER be exported, and thus can't be backuped.

Another usage of the key pair generated by this API is: you can save them to two files in hard disk and when your want to debug your code, you can import these files to virtual devices.

**Requirement:**

Hardware version: Senselock EL2.x

## A.1.2. R_GenerateRsaKeys

Generate a RSA key pair of PKCS#1 format on PC.

```
int WINAPI R_GenerateRsaKeys(
  R_RSA_PUBLIC_KEY *pPubKey,
  R_RSA_PRIVATE_KEY *pPriKey);
```

**Parameters:**

| | |
|---|---|
| *pPubKey* | [out] Pointer to `R_RSA_PUBLIC_KEY` struct,used to store the public key. |
| *pPriKey* | [out] Pointer to `R_RSA_PRIVATE_KEY` struct,used to store the private key. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

RSA key pair generated by this function can be used in the software version cryptographic algorithms. It can also be of the format defined in Senselock ELAPI header file(sense4.h) after some simple processing.

`R_RSA_PUBLIC_KEY` struct is defined as follows:

```
  typedef struct {
      unsigned int bits;
      unsigned char modulus[MAX_RSA_MODULUS_LEN];
      unsigned char exponent[MAX_RSA_MODULUS_LEN];
  } R_RSA_PUBLIC_KEY;
```

`R_RSA_PRIVATE_KEY` struct is defined as follows:

```
  typedef struct {
      unsigned int bits;
      unsigned char modulus[MAX_RSA_MODULUS_LEN];
      unsigned char publicExponent[MAX_RSA_MODULUS_LEN];
      unsigned char exponent[MAX_RSA_MODULUS_LEN];
      unsigned char prime[2][MAX_RSA_PRIME_LEN];
      unsigned char primeExponent[2][MAX_RSA_PRIME_LEN];
      unsigned char coefficient[MAX_RSA_PRIME_LEN];
  } R_RSA_PRIVATE_KEY;
```

Compared the two above structs with the structs of RSA key pair defined in header file *sense4.h* and you will find that the only difference between them lies on one struct member variable: `unsigned int bits.`

**Requirement:**

Hardware version: hardware-independent

## A.1.3. X_Pub2Cos

Convert a RSA public key of PKCS#1 format to the format supported in Senselock ELhardware.

```
int WINAPI X_Pub2Cos(
  COS_RSA_PUBLIC_KEY *pCosPubKey,
  R_RSA_PUBLIC_KEY *pPubKey);
```

**Parameters:**

| | |
|---|---|
| *pCosPubKey* | [out] Pointer to struct COS_RSA_PUBLIC_KEY, storing the public key of Senselock ELformat. |
| *pPubKey* | [in] Pointer to struct R_RSA_PUBLIC_KEY,storing the public key of PKCS#1 format. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

None

**Requirement:**

Hardware version: Senselock EL2.x

### A.1.4. X_Pri2Cos

Convert a RSA private key of PKCS#1 format to the format supported in Senselock ELhardware.

```
int WINAPI X_Pri2Cos(
  COS_RSA_PRIVATE_KEY *pCosPriKey,
  R_RSA_PRIVATE_KEY *pPriKey);
```

**Parameters:**

| | |
|---|---|
| *pCosPriKey* | [out] Pointer to struct `COS_RSA_PRIVATE_KEY`, storing the private key of Senselock ELformat. |
| *pPriKey* | [in] Pointer to struct `R_RSA_PRIVATE_KEY`,storing the private key of PKCS#1 format. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

None

**Requirement:**

Hardware version: Senselock EL2.x

### A.1.5. X_Cos2Pub

Convert a RSA public key of the format supported in Senselock ELhardware to PKCS#1 format.

```
int WINAPI X_Cos2Pub(
  R_RSA_PUBLIC_KEY *pPubKey,
  COS_RSA_PUBLIC_KEY *pCosPubKey);
```

**Parameters:**

| | |
|---|---|
| *pPubKey* | [out] Pointer to struct `R_RSA_PUBLIC_KEY`, storing the public key of PKCS#1 format. |
| *pCosPubKey* | [in] Pointer to struct `COS_RSA_PUBLIC_KEY`,storing the public key of Senselock ELformat. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

None

**Requirement:**

Hardware version: Senselock EL2.x

## A.1.6. X_Cos2Pri

Convert a RSA private key of the format supported in Senselock ELhardware to PKCS#1 format.

```
int WINAPI X_Cos2Pri(
  R_RSA_PRIVATE_KEY *pPriKey,
  COS_RSA_PRIVATE_KEY *pCosPriKey);
```

**Parameters:**

| | |
|---|---|
| *pPriKey* | [out] Pointer to struct R_RSA_PRIVATE_KEY, storing the private key of PKCS#1 format. |
| *pCosPriKey* | [in] Pointer to struct COS_RSA_PRIVATE_KEY,storing the private key of Senselock ELformat. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

None

**Requirement:**

Hardware version: Senselock EL2.x

## A.2. Cryptographic Algorithm Functions

### A.2.1. RSAPublicEncrypt

RSA public key encryption using PKCS#1 mode.

```
int WINAPI RSAPublicEncrypt(
  unsigned char *output,
  unsigned int *outputLen,
  unsigned char *input,
  unsigned int inputLen,
  R_RSA_PUBLIC_KEY *publicKey);
```

**Parameters:**

| | |
|---|---|
| *output* | [out] Output cipher. |
| *outputLen* | [out] Length of the output. It will be 128 if the RSA key is of 1024 bits. |
| *input* | [in] Input plaintext |
| *inputLen* | [in] Length of input. It can't exceed 117 if the RSA key is of 1024 bits. |
| *publicKey* | [in] Pointer to struct `R_RSA_PUBLIC_KEY`, the public key used for encryption. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

This function has completely the same PKCS encryption mode as Senselock ELSES `_rsa_enc()`.

**Requirement:**

Hardware version: hardware-independent

### A.2.2. RSAPrivateDecrypt

RSA private key decryption using PKCS#1 mode.

```
int WINAPI RSAPrivateDecrypt(
  unsigned char *output,
  unsigned int *outputLen,
  unsigned char *input,
  unsigned int inputLen,
  R_RSA_PRIVATE_KEY *privateKey);
```

**Parameters:**

| | |
|---|---|
| *output* | [out] Output plaintext. |
| *outputLen* | [out] Length of the output. It can't exceed 117 if the RSA key is of 1024 bits. |
| *input* | [in] Input cipher. |
| *inputLen* | [in] Length of input. It will be 128 if the RSA key is of 1024 bits. |
| *privateKey* | [in] Pointer to struct `R_RSA_PRIVATE_KEY`, the private key used for decryption. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

This function has completely the same PKCS encryption mode as Senselock ELSES `_rsa_dec()`.

**Requirement:**

Hardware version: hardware-independent

### A.2.3. Sign

RSA signing using PKCS#1 mode.

```
int WINAPI Sign(
  int digestAlgorithm,
  unsigned char *plain,
  unsigned int plainLen,
  unsigned char *signature,
  unsigned int *signatureLen,
  R_RSA_PRIVATE_KEY *privateKey);
```

**Parameters:**

| | |
|---|---|
| *digestAlgorithm* | [in] Hash algorithm used in signing. |
| *plain* | [in] Data to be signed |
| *plainLen* | [in] Length of the data to be signed. |
| *signature* | [out] Signature. |
| *signatureLen* | [in] Length of the signature. It will be 128 if the RSA key is of 1024 bits. |
| *privateKey* | [in] Pointer to struct `R_RSA_PRIVATE_KEY` the private key used for signing. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

This function has completely the same PKCS signing mode as Senselock ELSES `_rsa_sign()`.

**Requirement:**

Hardware version: hardware-independent

### A.2.4. Verify

RSA signature verification using PKCS#1 mode.

```
int WINAPI Verify(
  int digestAlgorithm,
  unsigned char *plain,
  unsigned int plainLen,
  unsigned char *signature,
  unsigned int signatureLen,
  R_RSA_PUBLIC_KEY *publicKey);
```

**Parameters:**

| | |
|---|---|
| *digestAlgorithm* | [in] Hash algorithm used in signing. |
| *plain* | [in] Original data |
| *plainLen* | [in] Length of the original data. |
| *signature* | [in] Signature. |
| *signatureLen* | [in] Length of the signature. It will be 128 if the RSA key is of 1024 bits. |
| *publicKey* | [in] Pointer to struct `R_RSA_PUIBLIC_KEY`, the public key used for signature verification. |

**Return values:**

Return RE_SUCCESS if verification succeeds or corresponding error code otherwise.

**Remarks:**

This API has not corresponding SensIV SES. The SES `_rsa_veri()` can't handle directly the original data and thus has minor difference with this API.

If you want to realize the same purpose using SES, you have to call `_sha1_xxx()` functions to complete hash calculation first and then call `_rsa_veri` (the second mode) to verify the signature.

**Requirement:**

Hardware version: hardware-independent

### A.2.5. Digest

Hash algorithms, including SHA1、MD5、MD2.

```
int WINAPI Digest(
  int digestAlgorithm,
  unsigned char *plain,
  unsigned int plainLen,
  unsigned char *digest,
  unsigned int *digestLen);
```

**Parameters:**

| | |
|---|---|
| *digestAlgorithm* | [in] Hash algorithm used: |
| | • `DA_MD2` MD2 |
| | • `DA_MD5` MD5 |
| | • `DA_SHS` SHA1 |
| *Plain* | [in] Message to be hashed. |
| *plainLen* | [in] Length of the message. |
| *Digest* | [out] Digest. |
| *digestLen* | [out] Length of the digest. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

None

**Requirement:**

Hardware version: hardware-independent

### A.2.6. DES

DES algorithm – ECB mode.

```
int WINAPI DES(
  unsigned char *key,
  int encrypt,
  unsigned char *output,
  unsigned char *input,
  unsigned int inputLen);
```

**Parameters:**

| | |
|---|---|
| *key* | [in] 8 octets secret key. |
| *encrypt* | [in] flag, 1 means encryption and 0 means decryption. |
| *output* | [out] output of encryption/decryption. |
| *input* | [in] input data. |
| *inputLen* | [in] Length of the input, must be multiple of 8. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

When used for encryption, this API is equal to SES _des_enc() and when used for decryption, it's equal to SES_des_dec() .

**Requirement:**

Hardware version: hardware-independent

### A.2.7. TDES

TDES algorithm in ECB mode.

```
int WINAPI TDES(
  unsigned char *key,
  int encrypt,
  unsigned char *output,
  unsigned char *input,
  unsigned int inputLen);
```

**Parameters:**

| | |
|---|---|
| *key* | [in] 16 octets secret key. |
| *encrypt* | [in] flag, 1 means encryption and 0 means decryption. |
| *output* | [out] output of encryption/decryption. |
| *input* | [in] input data. |
| *inputLen* | [in] Length of the input, must be multiple of 8. |

**Return values:**

Return RE_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

When used for encryption, this API is equal to SES `_tdes_enc()` and when used for decryption, it's equal to SES`_tdes_dec()`.

**Requirement:**

Hardware version: hardware-independent

## A.3. Error Code Index

| | | |
|---|---|---|
| RE_SUCCESS | 0x0000 | Operation successful. |
| RE_CONTENT_ENCODING | 0x0400 | Wrong content encoding. |
| RE_DATA | 0x0401 | Wrong data. |
| RE_DIGEST_ALGORITHM | 0x0402 | Invalid digest algorithm. |
| RE_ENCODING | 0x0403 | Wrong encoding. |
| RE_KEY | 0x0404 | Invalid key. |
| RE_KEY_ENCODING | 0x0405 | Wrong key encoding. |
| RE_LEN | 0x0406 | Wrong length. |
| RE_MODULUS_LEN | 0x0407 | Wrong Modulus length. |
| RE_NEED_RANDOM | 0x0408 | Random number needed. |
| RE_PRIVATE_KEY | 0x0409 | Error in private key encryption. |
| RE_PUBLIC_KEY | 0x040a | Error in public key decryption. |
| RE_SIGNATURE | 0x040b | Error in signing. |
| RE_SIGNATURE_ENCODING | 0x040c | Wrong signature encoding. |
| RE_ENCRYPTION_ALGORITHM | 0x040d | Wrong encryption algorithm. |

# Appendix B. Hardware Features

| Item | | Value | Description |
|---|---|---|---|
| USB Version | | 1.1 low speed | |
| CPU | | 16bits,16 MHz | |
| RAM (bytes) | VM Mode | 254 + 2047 | |
| | XA Mode | 2047 | |
| Memory (bytes) | | 8 ~ 64 K | |
| IO Buffer (bytes) | | 250 | Can also be used as RAM. For Senselock ELNet, this value is 244. |
| Working temperature | | -10 ~ 85 ℃ | |

# Appendix C. SES Index

# Appendix D. Driver Installation API Reference

Generally speaking, the driver installer tool provided by us can meet most of the needs. If you hope to customize your own driver installation procedure or GUI, you can use driver installation API. In fact, the driver installer tool provided by us is also developed based on this API.

Note: While using driver to install API, mkSetup.dll can not be invoked alone. Under the directory of mkSetup.dll, win98, winlh64, winlh86, winxp64, winxp86 directories and contents must be preserved unchanged, in fact, mkSetup executes thoses files to complete the installation/uninstallation.

## D.1.1. s4drv_GetDriverInfo

To get the information of drivers already installed in current system.

```
DWORD WINAPI s4drv_GetDriverInfo(PDRIVER_INFO pDrvInfo);
```

**Parameters:**

| | |
|---|---|
| *pDrvInfo* | [out] Pointer to `DRIVER_INFO` struct. |

**Return values:**

Return ERR_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

`DRIVER_INFO` struct holds the driver information of current system which is defined as follows:

```
typedef struct _DRIVER_VERSION
{
  DWORD DriverNum;
  DWORD Version[8];
} DRIVER_INFO, *PDRIVER_INFO;
```

Remarks for the struct member variables：

| | |
|---|---|
| *DriverNum* | Number of installed drivers in current system. For example, if there are a driver of version 2.0.0.0 and another driver of version 2.0.0.1 in current system, this return value will e 2. Commonly, this number is 1 or 0. |
| *Version* | Driver version. Every driver version has a corresponding array element. For example, if there has been a driver of version 2.0.1.0, then `Version[0]=0x02000100`. |

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

### D.1.2. s4drv_Install

To install device driver.

```
DWORD WINAPI s4drv_Install(
  LPCSTR lpszDestPath,
  DWORD dwCount,
  DWORD dwFlag);
```

**Parameters:**

| | |
|---|---|
| *lpszDestPath* | [in] Destination path of driver installation, and if the directory specified by this path doesn't exist, this function will create it automatically. |
| *dwCount* | [in] For Senselock ELdongle users, this parameter can just be ignored. If one needs to install PC/SC driver(*dwFlag* set to be DRV_FLAG_PCSC), he must specify this count of smart card reader(s)(1~8) |
| *dwFlag* | [in] Two flags supported currently:<br>• DRV_FLAG_PCSC install PC/SC driver<br>• DRV_FLAG_CLEAR_OLD remove old driver(s) (recommended) |

**Return values:**

Return ERR_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Sometimes, you may need to restart your system to complete the driver installation. To check if the system need reboot, please use `s4drv_IsNeedReboot()`.

After calling this function correctly, the system will copy all the files needed to the specified path. To complete driver installation, please reboot operating system(in case of need), and plug your device into computer; If you have already plugged your device before driver installation, please replug it.

It's strongly recommended that you remove the old version driver(s) when installing new driver, or it may lead to potential errors.
For details regarding PC/SC driver, please refer to corresponding material of PC/SC Group.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

**Sample:**

Please refer to the sample of `s4drv_IsNeedReboot()`.

### D.1.3. s4drv_Uninstall

To uninstall all the driver(s) already installed in current system.

```
DWORD WINAPI s4drv_Uninstall();
```

**Parameters:**

*None*

**Return values:**

Return ERR_SUCCESS if the function succeeds or corresponding error code otherwise.

**Remarks:**

Sometimes, system may need to be restarted to complete the driver uninstallation process. To check if the system need reboot, please use `s4drv_IsNeedReboot()`.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

**Sample:**

Please refer to the sample of `s4drv_IsNeedReboot()`.

## D.1.4. s4drv_IsNeedReboot

To check whether the operating system needs reboot.

```
BOOL WINAPI s4drv_IsNeedReboot();
```

**Parameters:**

*None*

**Return values:**

Return `TRUE` if the system needs reboot, `FALSE` otherwise.

**Remarks:**

To check whether or not the operating system needs reboot so as to complete the process of driver installatin/uninstallation.

**Requirement:**

Hardware version:Senselock EL2.x desktop version,network version

**Sample:**

```
/* This demonstrates how to install or uninstall Senselock
ELdevice driver using API. */

#include "stdio.h"
#include "stdlib.h"
#include "..\include\s4drv.h"

/* paramter
//  -u: uninstall
//  -nc: not clear old drivers
//  -pcsc: install PC/SC driver
//  -rd x: reader count(PC/SC only)
//  -path string: the dest path of driver file
*/
BOOL isInst = TRUE;
BOOL isPcsc = FALSE;
BOOL isClearOld = TRUE;
DWORD dwReaderCount = 1;
char lpszPath[MAX_PATH] = {0};
BOOL isNeedReboot = FALSE;

void HandleArgment(char* argv[], int argc)
{
  int i;
  for(i=1; i<argc; i++)
  {
    if(0 == strcmp(argv[i], "-u"))
      isInst = FALSE;
    else if(0 == strcmp(argv[i], "-pcsc"))
      isPcsc = TRUE;
    else if(0 == strcmp(argv[i], "-nc"))
      isClearOld = FALSE;
    else if(0 == strcmp(argv[i], "-rd"))
    {
      if(i+1 < argc)
        dwReaderCount = atoi(argv[i+1]);
      i++;
    }
    else if(0 == strcmp(argv[i], "-path"))
    {
      if(i+1 < argc)
        strcpy(lpszPath, argv[i+1]);
      i++;
    }
  }
}

int main(int argc, char* argv[])
{
  DWORD dwRet;
  char* pointer;
  GetSystemDirectory(lpszPath, MAX_PATH);
  pointer = strstr(lpszPath, ":\\");
  pointer[2] = '\0';

  // set default path;
  strcat(lpszPath, "Program Files\\Senselock\\Driver");

  HandleArgment(argv, argc);

  if(isInst)
  {
    DWORD dwFlag = 0;
    dwFlag |= (isClearOld ? DRV_FLAG_CLEAR_OLD : 0);
    dwFlag |= (isPcsc ? DRV_FLAG_PCSC : 0);
    printf("Start to install Senselock ELDrivers...");
    dwRet = s4drv_Install(lpszPath, dwReaderCount, dwFlag);
    if(dwRet != ERR_SUCCESS)
    {
      printf("Failed! Error: 0x%08x\n", dwRet);
      return 1;
    }
  }
  else
  {
    printf("Start to uninstall Senselock ELDrivers...");
    dwRet = s4drv_Uninstall();
    if(dwRet != ERR_SUCCESS)
    {
      printf("Failed! Error: 0x%08x\n", dwRet);
    }
  }
  printf("OK!\n");

  if(s4drv_IsNeedReboot())
    printf("You must restart your system!\n");

  return 0;

}
```