

ROCKY6 Smart Tutorial

Table of Contents

CHAPTER 1 OVERVIEW	1
1.1 BRIEF INTRODUCTION.....	1
1.2 ENCRYPTION REQUIREMENTS	1
1.3 COMMON ERRORS.....	1
CHAPTER2 KEIL INTEGRATED DEVELOPMENT ENVIRONMENT.....	4
2.1 KEIL IDE SETTINGS	4
2.2 CREATING A PROJECT	4
2.3 PROJECT OPTION SETTINGS	4
CHAPTER 3 ROCKEY6 SMART ESSENTIAL.....	8
3.1 DEVELOPMENT INTRODUCTION.....	8
3.2 SOFTWARE PROTECTION CONCEPTS.....	8
3.3 CREATING A C51 PROJECT	9
3.4 CREATING EXECUTABLE FILE IN THE DONGLE.....	10
3.5 EDITING AND ENCRYPTING DONGLE INTERCOMMUNICATION PROGRAM	11
3.6 CORE CODE SELECTION	13
3.7 CONCLUSION	13
CHAPTER4 IDE APPLICATION.....	15
4.1 FORMATTING A CARD.....	15
4.2 BURN IN BULK.....	15
4.3 PROGRAM DEBUGGING.....	16
CHAPTER 5 BASIC PROGRAMMING TECHNIQUES.....	18
5.1 FILE READING AND WRITING	18
5.2 FLOATING POINT COMPUTATION.....	20
5.3 ENCRYPTION AND DECRYPTION.....	23
5.4 FUNCTION EXPANSION.....	26
5.5 DONGLE IDENTIFICATION	26
5.6 MULTI-THREAD SUPPORT	27
CHAPTER 6 SOFTWARE RELEASE MANAGEMENT.....	28
6.1 USING TIMES LIMITATION.....	28
6.2 USING TIME CONTROL	28
6.3 FUNCTION LIMITATION.....	30
CHAPTER7 REMOTE UPDATE.....	34
7.1 MULTI-MODULE MANAGEMENT	34
7.2 USING REMOTE UPDATE TAG.....	35

7.3 REMOTE MODULE MANAGEMENT	36
------------------------------------	----

Chapter 1 Overview

1.1 Brief Introduction

ROCKEY6 Smart dongle is the 6th generation programmable dongle from FEITIAN Technologies. It could convert your core codes into an executable file and store the file inside the dongle. The executable file can only be executed inside the dongle. It is completely isolated from the vulnerable computer environments. This would prevent the executable file from being traced and copied. Therefore, the more complicated the code would be, the less possibilities the program is cracked.

With ROCKEY6 Smart remote update management features, the user could update the dongle software frequently and safely. It would remain dongle's safety and reliabilities as time goes. Based on those key facts, the ROCKEY6 Smart is also called "Uncrackable Dongle".

1.2 Encryption Requirements

ROCKEY6 Smart dongle key security features:

- (1) Unique - It is required to create and store an executable file inside the dongle. This inside file is completely sealed and isolated from the PC. Without the executable file inside the dongle, the entire program cannot be executed successfully or completely.
- (2) Inaccessibility - This dongle is based on the smart card chip. Its hardware is impossible to duplicate; and all data inside the chip cannot be accessed without proper authorities.

1.3 Common Errors

We have published a book named <<Software Encryption Theories and Applications>>. In this book, we have summarized some of the common errors for using dongle products. The following is some of the points.

1.3.1 Simple Search Mode

This mode is used to determine if the dongle is connected to PC. If the dongle exists, then the program is executed forward, and there would be not error messages prompted.

1.3.2 Memory Reading Distinction

When software developers use the memory dongle, this unsecured method is commonly used. Similarly, it is also commonly happened when the users use the stored data from the dongle.

By using this method, all data saved in the dongle memory would be read out. They are either simply compared with the results from the executing program or used directly. In this way, the dongle intercommunication interface would be unprotected exposed. It would cause the encrypted data to be traced and located. Once the secured data is found, it is possible for the data to be simulated and the software to be cracked.

We would suggest using the dongle with memory capacities in following ways. Firstly, “separately use the writing data and the reading data”; it means to use dongle storage as a special memory space of the executing program. In this special memory space, the concurrent running programs could mutually share and exchange all data resources. One place is for writing data; another place is for reading data. For example, we could create a random number to the program once it starts. Every data exchange would use this random number in the program. And data can be only transferred after the successful computation. In this way, the process of writing and reading data in the dongle would be separated. Additional, this method would enhance the dongle security. That is, the cracker still cannot decrypt the program even the encryption method is identified. Because there are too many complied source code need to be interpreted. In most case, the decryption would be failed due to easier destruction of address offset.

1.3.3 Obvious Application Interface

When software developers configure the dongle, for the purpose of applying the dongle verification code in different languages (such as VB and Delphi), the verification code is usually encapsulated in a DLL file with an external verification function.

1.3.4 Weight Too Much For Envelope Encryption

When using the dongle, most software developers favor to use the envelope encryption tool for executable files. It looks fast, secure and convenient. However, it is not always true when we view this from different perspective.

Envelope encryption directly encrypts the binary executable file. It is convenient for no adding any extra encryption algorithms. But, since envelope encryption is standard technology today and it is quite well known for everybody; it is possible to be decrypted by using many of existing envelope decryption tools.

All envelope encryption tools provided by the dongle manufacturer are working with the dongle API. Once the program is encrypted via the dongle API before the envelope encryption, all dongle API calling functions will be safely hidden. It would achieve much better security result.

In other words, if envelope encryption is the only used method, it will not protect the software as well as we supposed. Therefore, we highly recommended our users to use the dongle API encryption and envelope encryption together.

1.3.5 Regulated IO Computation

When software developers configure the dongle, their programs usually need IO communication with the dongle programs. In some cases, numbers of dongle outputs occur exactly after numbers of the dongle inputs. It looks secure since part of program requiring the output from the dongle and never showing in the PC. However, since all dongle inputs and outputs are regulated occurred, it is possible for the data to be simulated and the software to be cracked.

1.3.6 Improper Computation

It refers to the paired encryption computations are used for both dongle and the PC programs. In other words, a program is encrypted in the dongle and decrypted in the PC with the same or reversed computation method. Once the crackers figure out the computation patterns, it wouldn't take long for them to crack the software.

Chapter2 Keil Integrated Development Environment

Now, the current version of Keil only supports the Keil C51. Its IDE is Keil uVision2 that could be downloaded from <http://www.keil.com/>, or get its DEMO version from ROCKEY6 Smart SDK (/Support/Keil uVersion2). To use them, all users need to add the file headers and relevant libraries to the project. After that, all features could be performed once their relevant system functions are called.

2.1 KEIL IDE Settings

Before the first time to use KEIL, ROCKEY6 Smart needs to be configured some project settings. Firstly, copy ‘\TOOLS\DEBUGER\RySSimulator.dll’ and ‘DIC32R.DLL’ from ROCKEY6 Smart SDK to ‘\C51\BIN’ and ‘\UV2’ in the ‘KEIL’ directory accordingly. And update the ‘TOOL.INI’ file. Then add “TDRV4=BIN\RySSimulator.dll (“FEITIAN ROCKEYSmart Simulator”)” to the C51 part. ‘TNRV4’ refers to a serial number. If this number is used by a program, then the next number would be used.

2.2 Creating a Project

To create a new project, click ‘Project’ -> ‘New Project’ from KEIL UV2. When a message box pops up, input project name and save it. Once project name is chosen to save, a window “Options for Target ‘Target1’” shows. Please select ‘CPU’ from the ‘51 series’; especially for those ‘CPU’s aren’t selected for ‘51 series’ at the first time. See Figure 2-1 below.

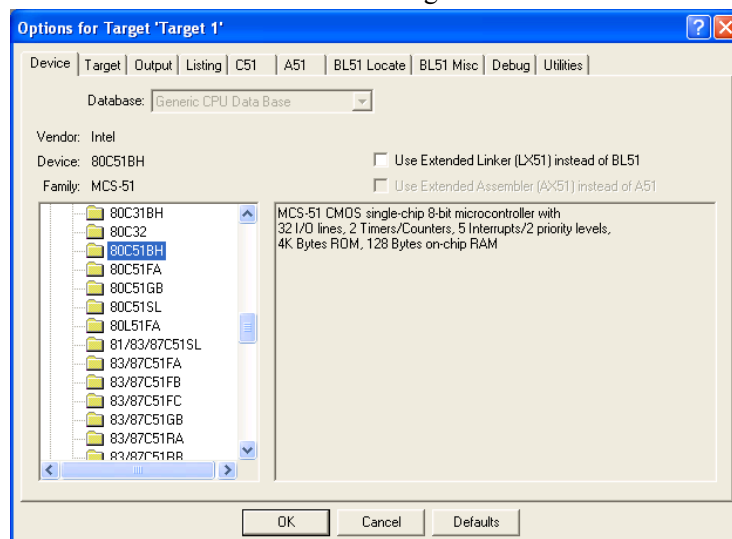


Figure 2-1

2.3 Project Option Settings

Open menu “Project -> Options for Target ‘Target1’”. For ‘Memory Model’ in ‘Target’ page, we could choose default ‘Small’. We suggest using this model simple because it usually executes efficiently. The other two models could also be used to their related libraries. However, since ‘Compact’ model settings are complicated, we would not suggest the new users to use it. Please

see figure 2-2.

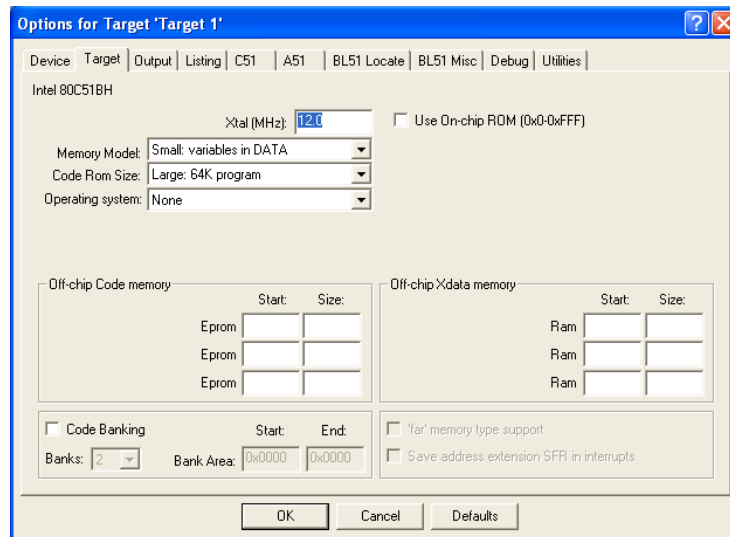


Figure 2-2

In the window of “Project -> Options for Target ‘Target1’”, choose “Create Hex file” from the ‘output’ page. And choose ‘Run User Program #1’, then input “hexhin.exe test.hex test.bin”. The project names may change corresponding to ‘test.hex’ and ‘test.bin’. Please copy ‘hexbin.exe’ from the SDK to your project directory.

NOTE: it is not necessary to copy it, if you only want to debug the program. But it is mandatory to do so, if you are going to burn the program in bulk to the dongle. The reason for that is for the process being designed to use the BIN file. Please see Figure 2-3 below.

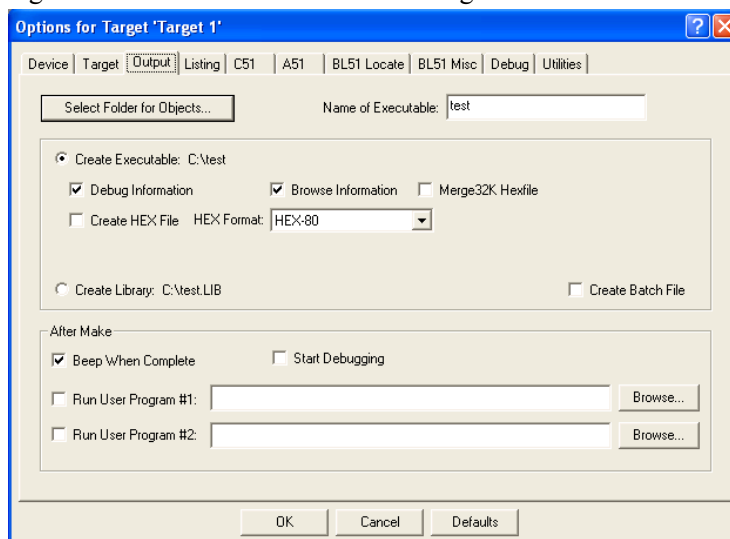


Figure 2-3

Please choose to use the simulator from the ‘DEBUG’ menu. Then choose “Go till main”. When program the ROCKEY6 Smart dongle, if we want to use the debugging program in Keil IDE, the setting is required.

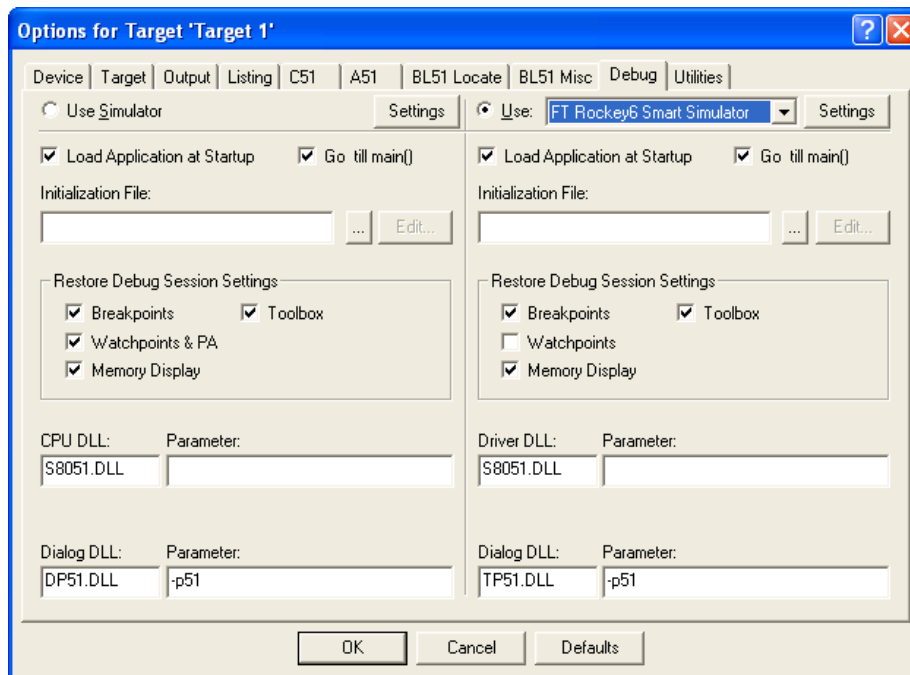


Figure 2-4

Click “Settings” button to configure the debugger. If a real card is connected, then choose “Use the Real Card”. Whether to select to use the real device depends on if you want to burn the program to the real card. If you don’t really need the real device, you could omit this step and debug your program directly with the simulator.

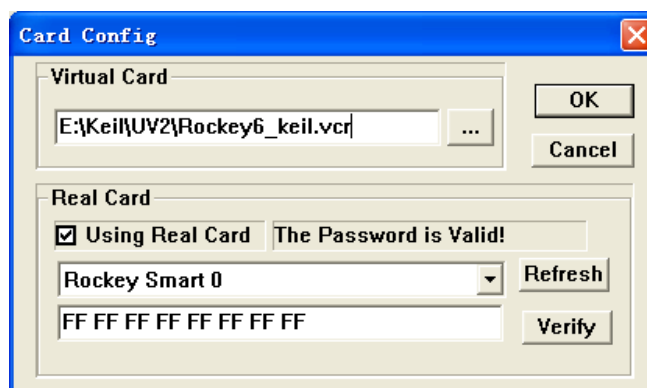



Figure 2-5

Finally, choose simulator from the “Utilities”. This simulator is used for download. After setting this, a button  would be showed in the KEIL main window. This button is used to download data to the card once program-debugging finishes.

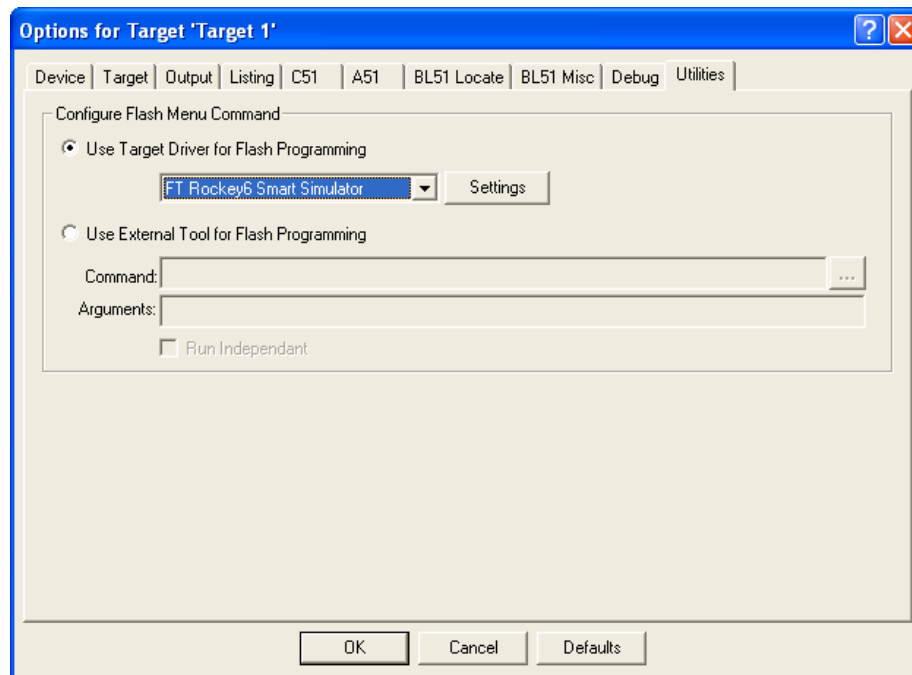


Figure 2-6

Choose “View -> Project Window” from the menu. In the left hand side of your window, once all default directory tree is expanded, choose “Source Group1”, and right click the mouse to select “Add files to Group ‘Source Group1’”. Then import the file header and libraries from ‘API32\C51’ into your project as showed in Figure 2-7 below.

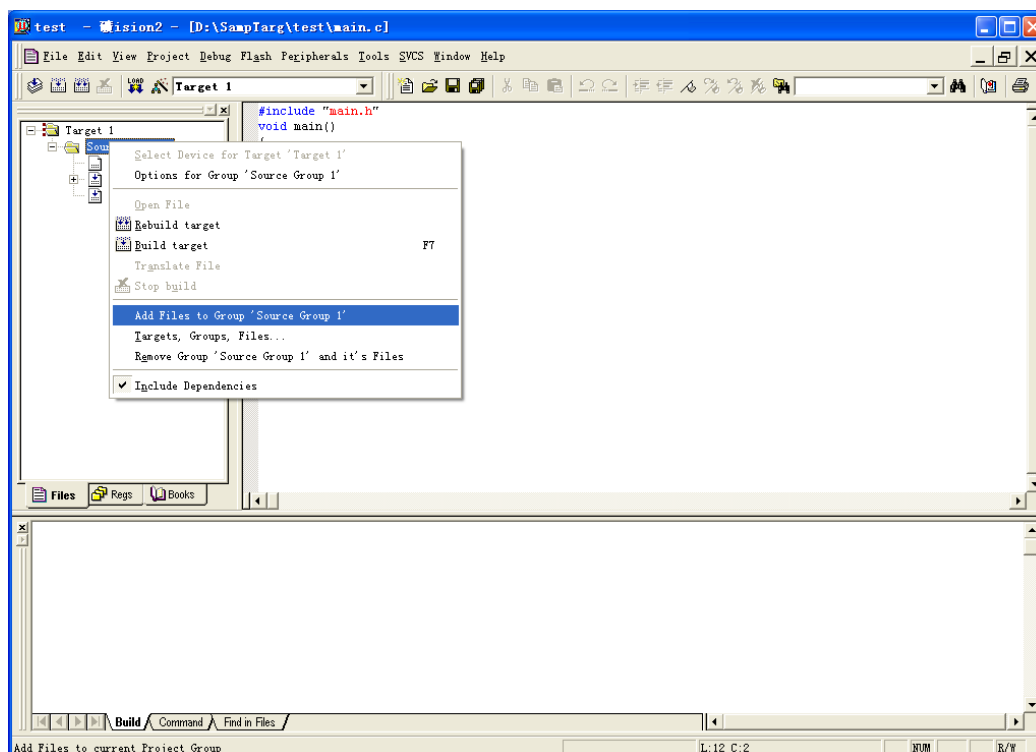


Figure 2-7

Chapter 3 ROCKEY6 Smart Essential

3.1 Development Introduction

ROCKEY6 Smart is a programmable dongle. It has two development models: one is dongle internal program; another is the external program communicated with the program inside dongle. After the dongle software and hardware get ready, please follow the steps below to develop a simple protection program.

3.2 Software Protection Concepts

ROCKEY6 Smart dongle is developed by using C language. The developers need to study the provided system functions usages before they deploy the dongle.

The following diagram is the process for dongle configuration:

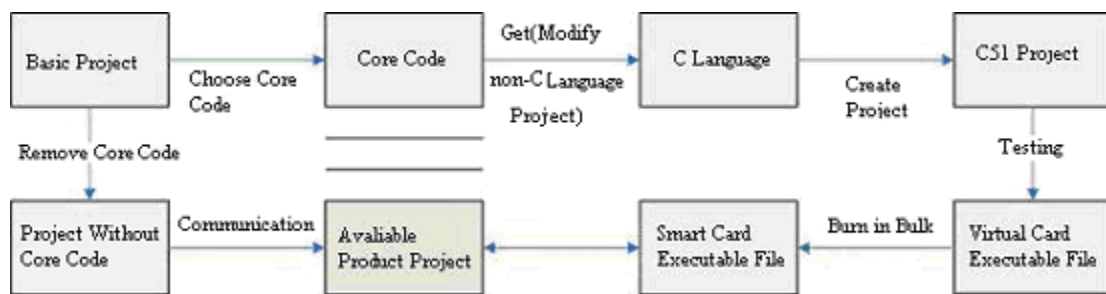


Figure 3-1 Development Process

Please consider the following algorithm: Input 15 bytes ID numbers, its output are 18 bytes numbers. This is a simply message-digest algorithm. Assume those number is a part of critical code, then we could save those numbers into the dongle. An example in C language is below:

```
1 unsigned char Wi[18] = {7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2,1};
2 char Ai[11]={'1','0','x','9','8','7','6','5','4','3','2'};
3 void ConvertID(char ID[15],char newID[18])
4 {
5     int i,j,s;
6     s=0;
7     memcpy(newID,ID,6);
8     newID[6]='1';
9     newID[7]='9';
10    memcpy(newID+8,ID+6,9);
11    for(i=0;i<17;i++)
12    {
13        j=(newID[i]-48)*Wi[i];
14        s+=j;
15    }
16    s%=11;
```

```

17     newID[17]=Ai[s];
18 }

```

Below is written in C51 C language. It would do the same thing.

```

1.  unsigned char Wi[18] = {7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2,1};
2.  char Ai[11]={'1','0','x','9','8','7','6','5','4','3','2'};
3.  void main(void)
4.  {
5.  int i,j,s;
6.  s=0;
7.  byte ID[15],newID[18];
8.  get_input(ID,0,0,15); //Input ID numbers
9.  memcpy(newID,ID,6);
10. newID[6]='1';
11. newID[7]='9';
12. memcpy(newID+8,ID+6,9);
13. for(i=0;i<17;i++)
14. {
15. j=(newID[i]-48)*Wi[i];
16. s+=j;
17. }
18. s%=11;
19. newID[17]=Ai[s];
20. set_response(newID, 18); //Output new ID numbers
21. exit(); //program ends
22. }

```

The 3^d line is the main function definition. The 8th line: 'get_input(ID,0,0,15)' is used for receiving the external input data. ID is buffer, parameter 2 means offset, 3 means the input data type, 0 means byte and 15 means data length. The 20th line: set_response(newID, 18) returns the data to the host machine. In this function, 'newID' refers to the buffer address, '18' means the length of the output data. 'exit()' means programming ending.

In KEIL compile environment, it uses C language. However, it is different with the standard C language due to its distinguished methods for defining variables. An example is: 'unsigned char xdata buf[128]'. 'xdata' means putting the variable into 'xdata' area. If a larger array is defined, it is usually assigned at 'xdata' area in order to get enough memory space. If you want to know more about C51 features, please refer to the C51 user manual for details.

3.3 Creating a C51 Project

Converting the core code into KEIL C language, and creating a C51 project. Then this part of core code could be compiled, debugged and downloaded to the dongle.

After creating a C51 project, please add your source code, SDK files:

'API32\C51\Small_Mode.lib' and 'sys_api.h' into the project.

Once you have done those, you could start to debug, compile and/or download your program.

3.4 Creating Executable File in The Dongle

Once the project is created, the source code could be compiled and debugged. It could be debugged in different methods, such as in steps or in blocks. See Figure 3-3:

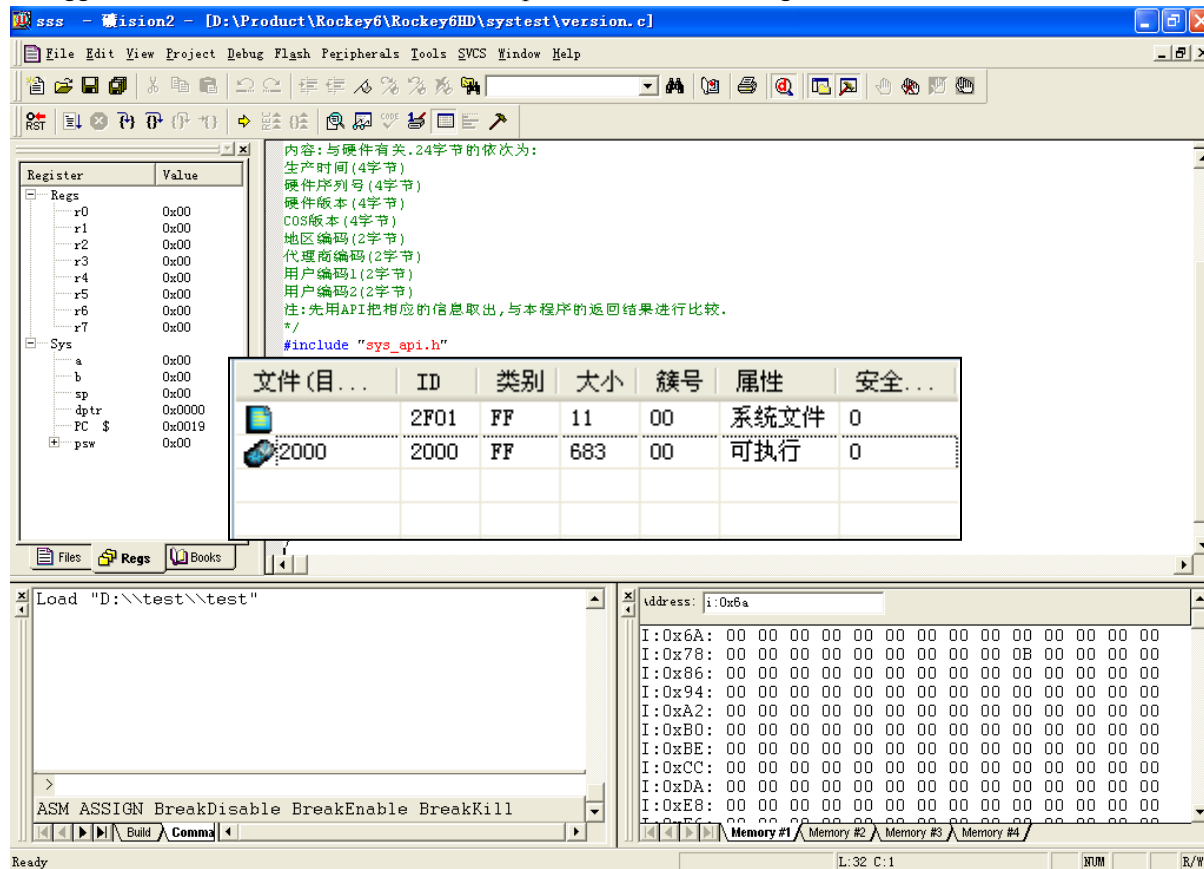


Figure 3-2 Debugging Window

If debugger meets function 'get_input' and/or 'set_response', it will pup up the following window for the data inputting in or data content displaying. (The example will use 'getversion' function).

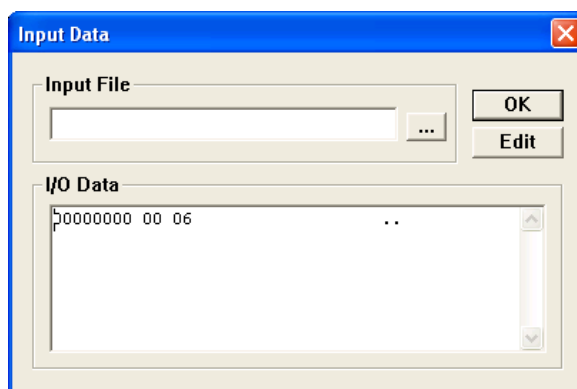


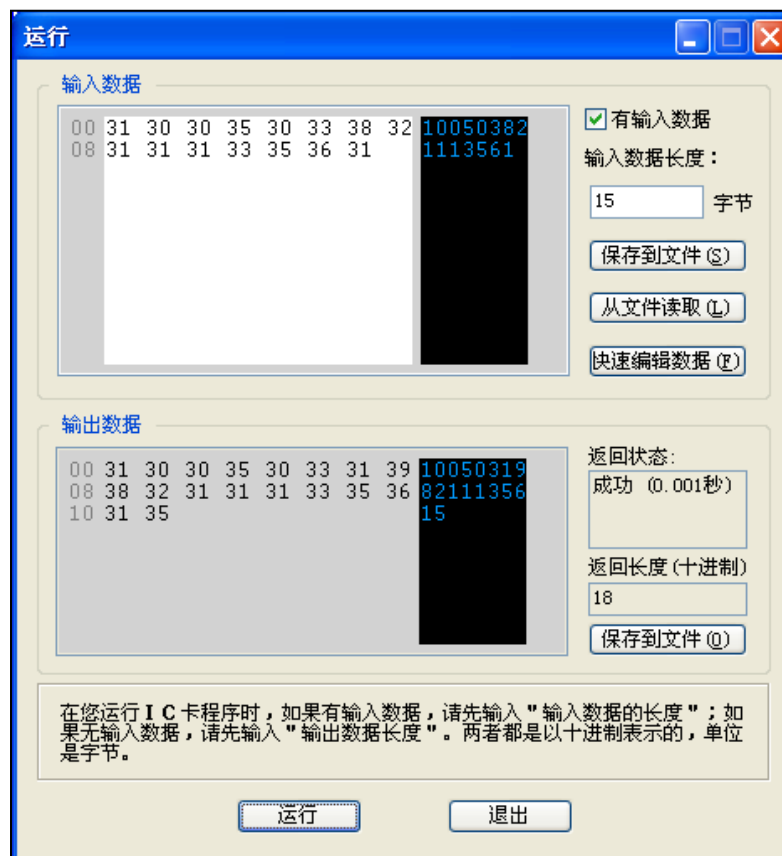
Figure 3-3 Output Data

Once the code is successfully compiled, it could be downloaded to the card by using the discussed method in previous section. Now, if you open IDE tool to browse the real devices, you could find a file with its name and ID are 2000. When the file is executed, it could dynamically fetch the files

it needed, and return the corresponded results.

文件(目...	ID	类别	大小	簇号	属性	安全...
	2F01	FF	11	00	系统文件	0
 2000	2000	FF	683	00	可执行	0

Figure 3-4 Files In The Real Card



运行

输入数据

00 31 30 30 35 30 33 38 32 10050382
08 31 31 31 33 35 36 31 1113561

☒ 有输入数据
输入数据长度：
15 字节
保存到文件(S)
从文件读取(L)
快速编辑数据(E)

输出数据

00 31 30 30 35 30 33 31 39 10050319
08 38 32 31 31 31 33 35 36 82111356
10 31 35 15

返回状态：
成功 (0.001秒)
返回长度(十进制)
18
保存到文件(O)

在您运行 I C 卡程序时，如果有输入数据，请先输入“输入数据的长度”；如果无输入数据，请先输入“输出数据长度”。两者都是以十进制表示的，单位是字节。

运行 退出

Figure 3-5 Simulated Executions

Now, click “Browse Real Device” button from the tool bar, then you could find the file. Once you have finished all previous steps, a communication module needs to be created for communicating to the card.

3.5 Editing and Encrypting Dongle Intercommunication Program

The steps for executing a dongle file:

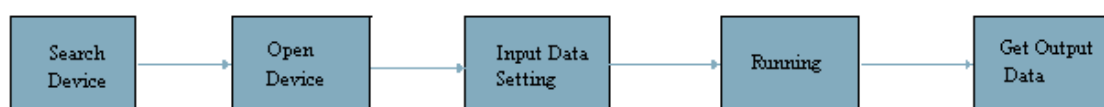


Figure 3-6 File Execution Steps

Below is configured API that is used for communications between application programs and ROCKEY6 Smart dongle:

```
EXTERN_C int WINAPI DIC_Find(DWORD UID);
The function is used for dongles attached to PC based on the user code, and returns the total numbers of attached dongle devices.
EXTERN_C int WINAPI DIC_Open(int hic, char* reader_name);
Open a dongle from its index, and return the handler of the dongle
EXTERN_C int WINAPI DIC_Command(int hic, int cmd, void* cmddata);
Send commands to the dongle. The first parameter is the handler of the dongle; the second parameter is command macro; the third parameter is the inputting/outputting structures for command macro
```

DIC_Set and DIC_Get are described in details at <<User Manual>>. Please note that those two functions are not used for any dongle operations. They are simply used to set a buffer. The address of the buffer would be transferred to the 3rd parameter of the function DIC_Command. This buffer is the needed structure. For all 'struct' supported C languages are not necessary to use those two functions. According to the definition of the function DIC_Command, those languages could directly work on the various structures based on different macros. The purposes of those two functions are to help the users to configure complicated data structures; especially those languages are not supporting the 'struct' such as VB etc.

Below is an example converting the 'ConvertID' to the function that could communicate with dongle:

```
1 void ConvertID(const char ID[15],char newID[18])
2 {
3     DICST_Before_Run_Data *bD=(DICST_Before_Run_Data *)new char[48];
4     DICST_After_Run_Data *aD=(DICST_After_Run_Data *)bD;
5     int count=DIC_Find();//Search dongle
6     int hic=0;
7     for(int i=0;i<count;i++)
8     {
9         if((hic=DIC_Open(i,NULL))>=0)
10             break;
11     }
12     if(hic==count)
13         return; //Card is not found
14     bD->RunID=0x2000;// Set the executable file name
15     bD->ParaSize=15;//Input the length of buffer
16     memcpy(bD->Para,ID,15);// Input buffer
17     int ret=DIC_Command(hic,RUN,bD);//Run the file and wait for the return data
18     memcpy(newID,aD->Result,aD->ResultSize);// Get return data
```

Now, the entire project protection work is done. The newly formed interface is the same as the previous one, and it doesn't need to be modified for execution.

3.6 Core Code Selection

From above example, we have introduced a fundamental software encryption protection method. This example is quite simple and used for describing the idea only. Please do not use it directly in your protection. Based on our data, lots of users are using dongles for only searching if they are attached to the PC, or only store a piece of dispensable code inside the dongle. Dongles used in these ways could not protect your software well. We would recommend our users to take more time for configure their dongles. Below are some important principles and methods used for dongle applications.

1. Code inside the dongle has to be vital.

This part of code has to be saved inside the dongle. Nobody could get it without the valid verification procedures. The main program could not be executed entirely and successfully without this vital part of code. As a result, even some crackers attempt to avoid this piece of code, the whole program cannot be executed successfully.

2. Try to choose the specific code that related to your application

Since the popular algorithms are well known, the software crackers could easily 'guess' the functionalities of the code and simulating them to crack the software. Based on a fact that most our clients are the skilled software programmers in their own software applications, we would recommend our clients to add their own special codes in the dongle. It is impossible for a cracker to know all the codes from various application fields. Thus, the dongle with the special code would be too hard to be broken for a cracker. For example, a graphic programmer could use part of code from the graphic processing software in the dongle; a PC games programmer could use part of code from the artificial intelligence algorithms; a mechanical engineer could use part of code from the mechanics algorithms etc. In this way, the dongles would have dynamical diversity for its security algorithm applications. It would make the cracking job too hard to accomplish.

3. From the performance aspect, do not give the heavy work to the dongle.

The code inside the dongle would be executed within the dongle. Since the speed of dongle microprocessor is slower than the PC CPU, from this point, dongle usually is a bottleneck for entire computation. Therefore, do not input the dongle code to an executing game, 'ONMOUSEMOVE' of the interface program etc. Please try to use this piece of code safely. Once it is correctly used, even with some complex computation, the whole system might be slow but won't be crashed.

3.7 Conclusion

From above, we have discussed the fundamental methods for using the software protection dongle.

Moreover, we also showed how to separate and convert the code into C language. Finally, we described how to arrange the main program and communication modules of the smart card, as well as how to choose core code in the dongle.

Again, please take more time for programming your dongle software. It would be crucial for protecting your software.

Chapter4 IDE Application

IDE is also a tool for card settings and program burning in bulk. In this chapter, the common functions of IDE would be introduced. If you need further information about IDE, please refer to the <<User Manual>> and <<Advanced User Manual>>.

4.1 Formatting a card

Formatting is used for initializing the system and clearing the existing data in a card. In ROCKEY6 Smart, format could be also used for resetting the extensive libraries. Once the connected real card is found, click “Card Operation -> Format” in the main menu. A pop-up window would be showed as the Figure 4-1 below. In the window, the users could fill in and update the volume and manufacture information. If the ROCKEY6 Smart expansive libraries need to be imported, please select relevant items in the window. The last thing is click [OK] to start formatting.

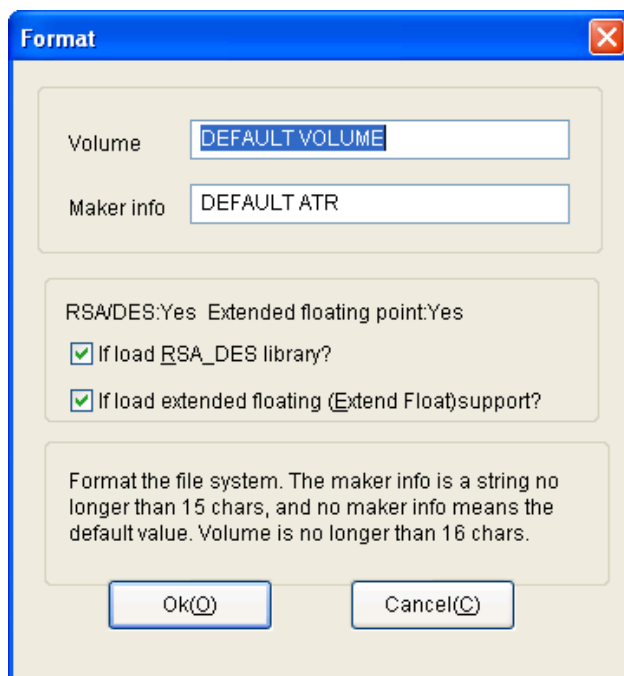


Figure 4-1 Formatting

4.2 Burn in Bulk

“Burn in Bulk” is usually used after the dongle protection programs are finished and just before the dongle release date. It will import all files of the virtual card into the real card. Once switch to the virtual device window, please choose “Selected Burning” for the intended files from the virtual card. Then click “Burn to The Real Card” from the “Card Operation” menu. A pup-up window will be showed as the Figure 4-2 below:

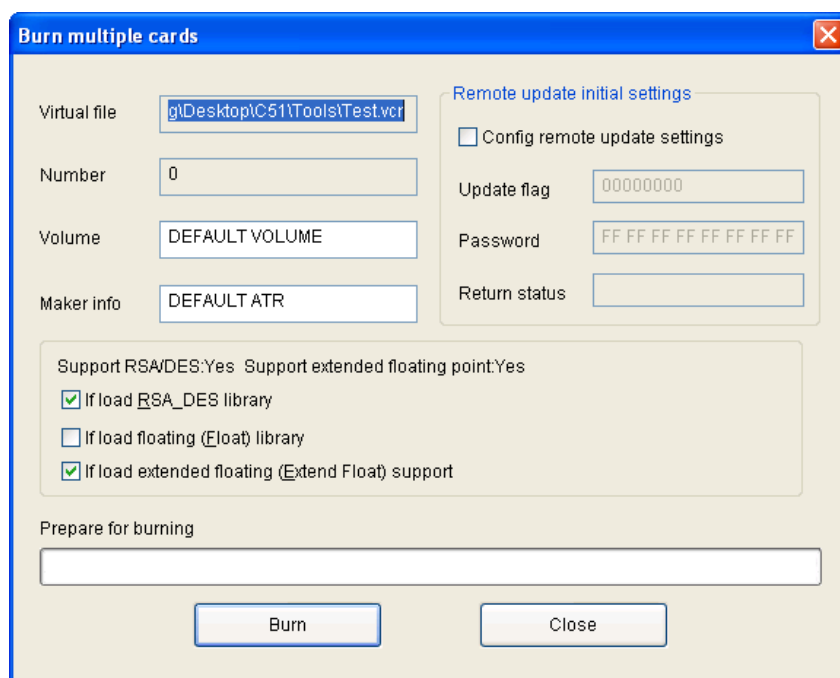


Figure 4-2 Burn in Bulk

‘Burn in Bulk’ could be also used for formatting the card. Therefore, it is also necessary to set “Volume” and Manufacturer Information” and the extension libraries for the ROCKEY6 Smart dongle. Its configuration is the same as the formatting. Additional, ‘Burn in Bulk’ could be also set the remote update tag. For the remote update tag usages, we will describe it later. Users could also find it from <<User Manual>>.

4.3 Program Debugging

Debugging is a basic function for the software development. Keil also supports powerful debugging functions, such as ‘break’ and ‘step’ etc. Debugging in C51 project, please use tool bar “Break” button (F9), then use tool bar “Run” (F5). After those two steps, the program debugging would be started. It is showed as Figure 4-3 below:

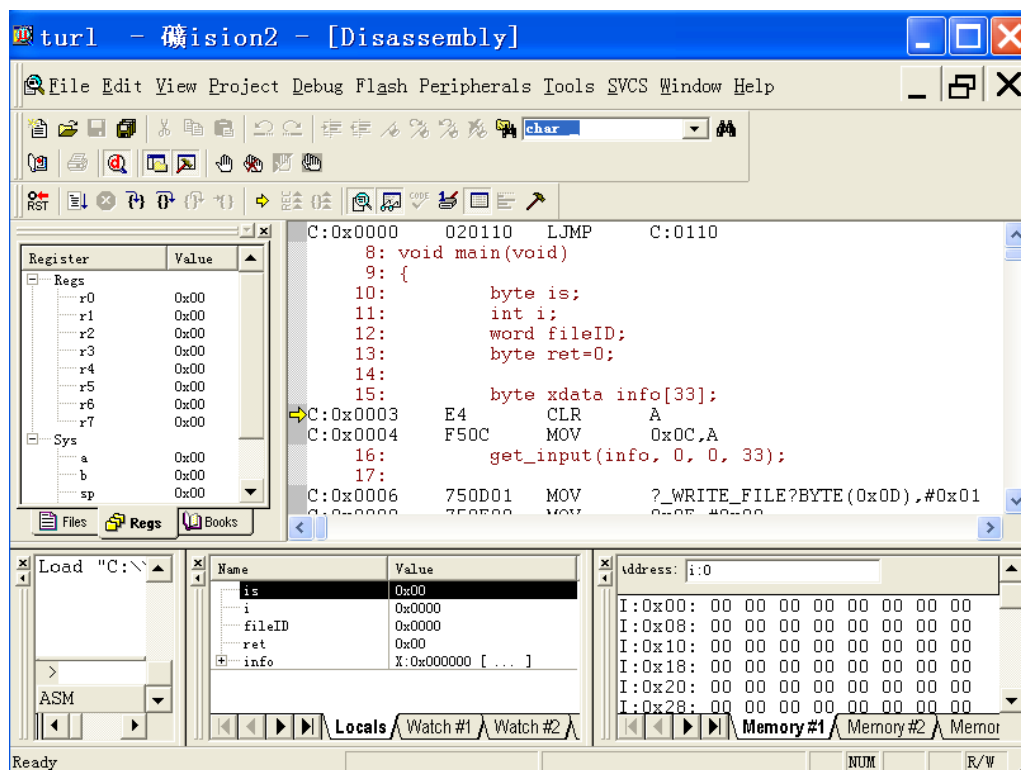


Figure 4-3 Debugging

In the process of debugging, the data in the register and RAM will be displayed. The 'run in step' function is also available at this time (F10) (F11).

Chapter 5 Basic Programming Techniques

In this chapter, we will introduce the basic programming techniques for ROCKEY6 Smart. It includes card internal files processing, double precision floating-point calculation, RSA and DES computation etc.

5.1 File Reading and Writing

File reading and writing are the basic techniques for programming a dongle. If there are huge amount of data, they need to be handled efficiently. There are two types of files in the ROCKEY6 Smart, data file and executable file. The executable file is the downloaded file inside the dongle (such as the file '2000' in previous chapter). It could be executed by the COS. In contrast, the data file includes normal file and internal file. The normal file could be read out and written in. It is usually accessed by the external application program via the API or card program via system calls. However, the internal file could be only accessed by the card internal executable files. Below is the files access control diagram. In most cases, the external program accesses the normal files after successfully password verification is only used for management. At the actual software protection project, the application program accesses the internal files mainly via the internal executable files. The access authorities are managed by the card executable files.

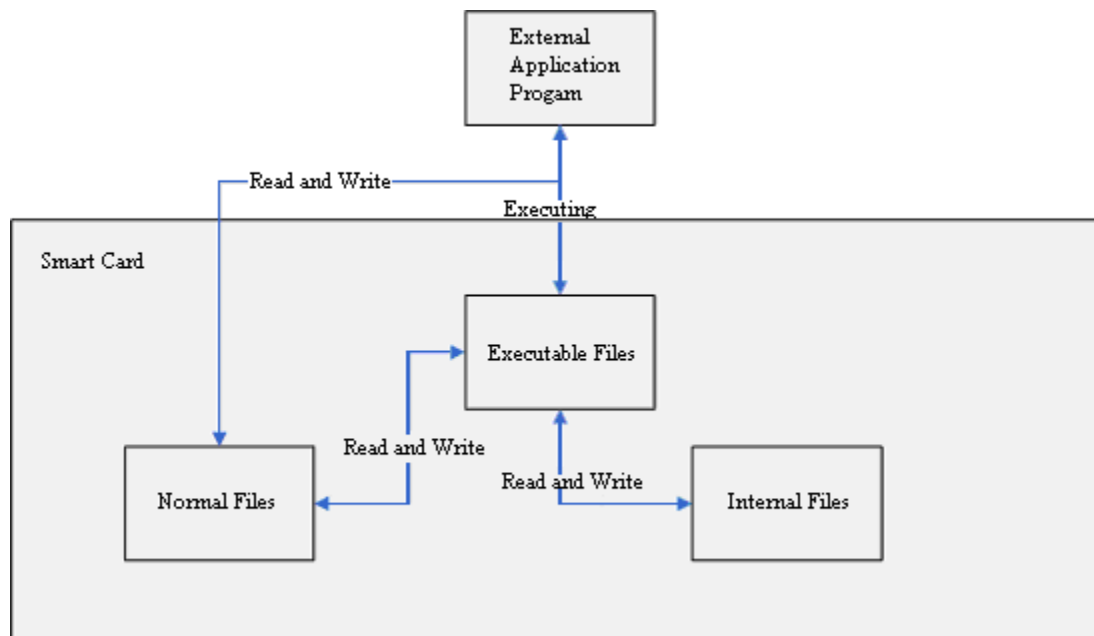


Figure 5-1

Now, we will describe the methods of file reading and writing in C51.

1. Creating file – The system call for creating file could be performed by 'create_file()'. Its prototype is showed below:

```
unsigned char creat_file(unsigned short wFileID,unsigned char pbFileName,unsigned char bAttrib, unsigned short wSize);
```

wFileID: File/Directory ID
bAttri : File/Directory attributes and their security levels
wSize : File length (counted in byte, if it is directory, the parameter is 0) parameter
pbFileName: File/Directory name buffer
(could set to 0, it means not to use the file name)
Return: Return code

Below is the simple code for creating file:

```
void main()
{
    byte ret;//Define word type variable. It is used to hold the return code.
    ret = create_file(0x1002, "1002",0x80,1024);
    // create file with ID to 1002, internal security level to 0,
    //size to 1024 bytes and name to 1002
    set_response(1,&ret); //output the return value for CreateFile
exit();
}
```

In this part of code, a file with ID to 1002 is created. Alternatively, the file ID, size and the other information could set to variables.

2. Reading file – reading file is performed by system calling ‘read_file()’. The definition of ‘read_file()’ is defined as follows:

unsigned char read_file(unsigned short offset, unsigned char *pd,unsigned short length);
offset: Offset position for the data to be read
pd: the buffer pointer, the place data stored
length: The length of the reading data
return: Return code

Additional, before reading and writing a file, please open file first. The system call for opening a file is ‘open_file’. its definition is showed below:

unsigned char open_file(unsigned char *pd,unsigned char open_mode);
pd: the pointer points to the file name or file ID
open_mode: open mode
00 indexing by file/directory ID (points to file/directory ID, the lower bytes is in the left)
01 indexing by file/directory names (points to file/directory name)
02 directly open the upper directory. If now ‘pd’ were NULL or other values, it would be ignored.
return: return code

The following methods could be also used to open and read a file:

```
#include "sys_api.h"
#include <string.h>
```

```

void main(void)
{
    byte xdata is;
    word xdata fileID;
    byte xdata result[50];
    byte xdata info[2];

    get_input(info,0, 0,2);
    memcpy(&fileID, info, 2);
    is = open_file((byte*)&fileID, 0); //Open file by the ID
    is = read_file(80, result,20); //Reading 20 bytes from the file offset position 80
    set_response(20, result);
    exit();
}

```

3. Writing file – the system call for writing file is ‘write_file’. Its function definition and usage are similar with ‘read_file()’. Please refer to <<Advanced User Manual>>. It is omitted here.
4. Deleting file – the system call for deleting file is ‘delete_file’. This function doesn’t contain any parameters. It is used for deleting current file. Firstly, open the file to be deleted. Then call this function to delete the file. The following code will perform the deletion according to the inputted ID.

```

void main()
{
    word FileID;
    byte Ret;
    get_input(&FileID,0, 1,1);

    Ret = open_file((byte*)&FileID,0);
    if(Ret!=0)
        exit();
    Ret = delete_file();
}

```

5.2 Floating Point Computation

ROCKEY6 Smart dongle provides hardware accelerated double precision floating-point computation. The variables in ‘double’ type cannot be processed in adding, subtracting, multiplying and dividing operations. They can be only operated in floating point functions. For example, in double type, “c = a + b” must write to “c = double_add(a,b)”. The floating-point functions for adding, subtracting, multiplying and dividing are listed below:

```

unsigned char double_add(unsigned char * a1,unsigned char * b1,unsigned char *result);
unsigned char double_sub(unsigned char * a1,unsigned char * b1,unsigned char *result);

```

```
unsigned char double_mul(unsigned char * a1,unsigned char * b1,unsigned char *result);
unsigned char double_div(unsigned char * a1,unsigned char * b1,unsigned char *result);
```

Besides the basic operations like adding, subtracting, multiplying and dividing, the floating-point calculation system also includes trigonometric functions, anti-trigonometric functions, square root functions, logarithmic functions and other commonly used mathematic functions. For their details, please refer to the Appendix of <<Advanced User Manual>>. Additional, before using the function, please make sure that the following two libraries are imported into the dongle (if you don't use the floating-point libraries, then importing those two libraries are optional.) – floating point library and extensive floating-point library.

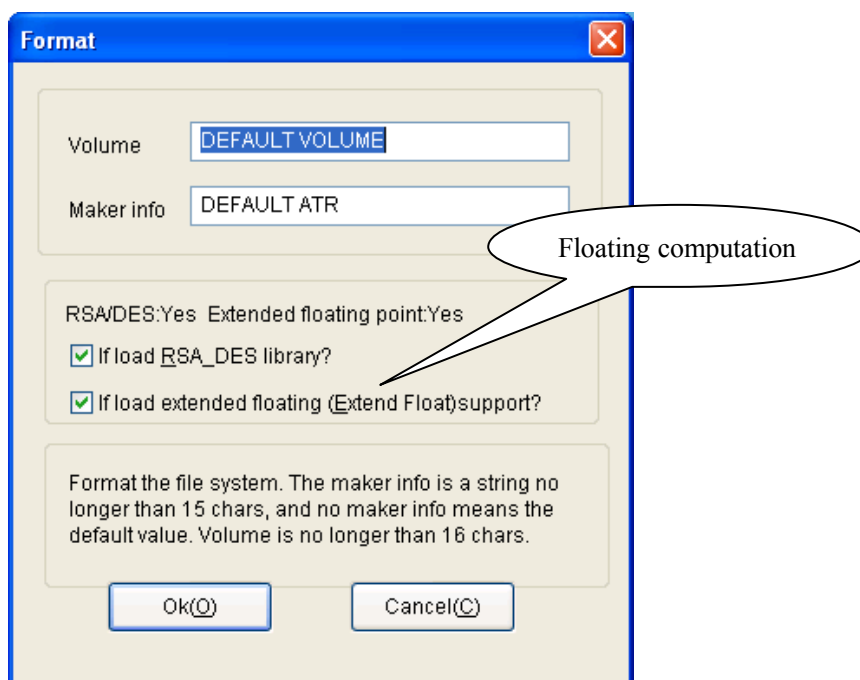


Figure 5-2

The following is an example of floating point multiplication by using the dongle:

```
void main()
{
    byte Ret;
    byte a[8];
    byte b[8];
    byte c[8]; //Used for output floating point c
    byte input[16];

    get_input(input, 0, 0, 16);
    memcpy(a, input, 8);          //input a
    memcpy(b, input+8, 8); //input b
```

```

Ret = double_mul(a, b, c);
if(Ret!=0){
    set_response(1,&Ret);
    exit();
}

set_response(8,c);
exit();
}

Using this function in the main program:
double double_mul(double a,double b)
{
    double c;
    DICST_Before_Run_Data *bD=(DICST_Before_Run_Data *)new char[24];
    DICST_After_Run_Data *aD=(DICST_After_Run_Data *)bD;
    int count=DIC_Find();
    int hic=0;
    for(int i=0;i<count;i++)
    {
        if((hic=DIC_Open(i,NULL))>=0)
            break;
    }
    if(hic==count) return 0;
    bD->RunID=0x2008;//assign the file ID used for floating point computation
    bD->ParaSize=16;//input 16 for the buffer size
    memcpy(bD->Para,&a,8);//set the first floating point number
    memcpy(bD->Para+8,&b,8);//set the second floating point number
    int ret=DIC_Command(hic,RUN,bD);//Run
    memcpy(&c,bD->Result,8);//get the result
    return c;
}

```

The following is the execution result for the program. The result of the dongle multiplication would be the same as the PC:

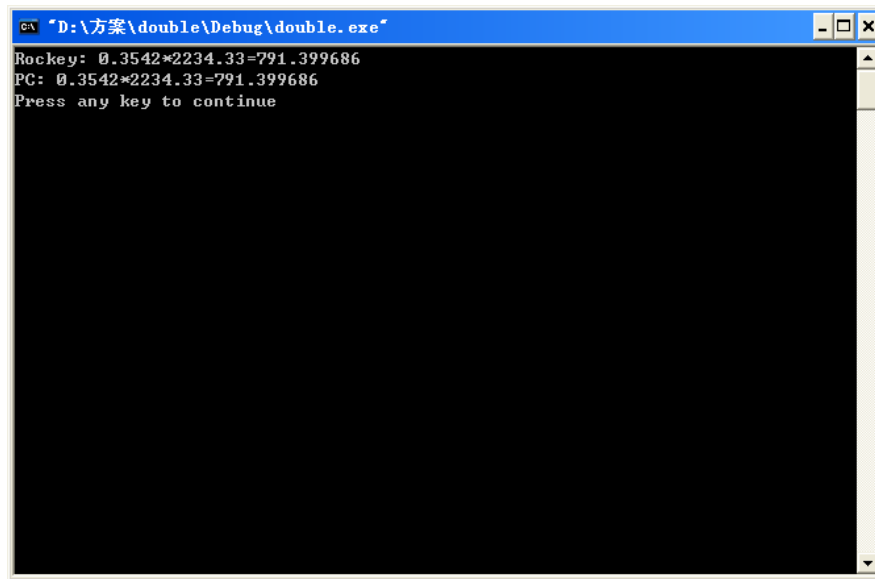


Figure 5-3

5.3 Encryption and Decryption

Encryption and decryption are the ROCKEY6 Smart internal algorithms by using the hardware microprocessor computation. This would get better execution efficiency, correct calculation and prevention of user tracking and debugging.

5.3.1 DES Encryption

DES encryption needs to save the keys in the card file before doing anything else. A key file could include many keys. When encrypting, different parameters could use distinguished keys.

Encryption - when encryption, the program inside the card could call 'des_enc' function to perform encryption. The application programs may also call 'DIC_Command' and transfer a 'DESENC' macro to perform encryption.

Decryption – when decryption, the program inside the card could call 'des_dec' function to perform decryption. The application program may also call 'DIC_Comman' and transfer a 'DESDEC' macro to perform decryption.

To use DES encryption and decryption method, a key file needs to be created firstly in the dongle. The layout for the key file is: every key is 10 bytes long; in which, key ID is 1 byte (8 bits), the parameter of the key length is 1 byte (8 bits), the key is 8 bytes. In this way, when the DES key file is created, it is assigned 10 bytes as one unit each time. When encryption, system call and/or API need to set a key ID as a parameter. The key ID is used for determining which key is chosen. For example, the following code will write two keys to a created key file. The key ID is 01 and 02.

```
memcpy(data, "\x01\x08\x01\x02\x03\x04\x05\x06\x07\x08", 10);
//data is a temporary buffer area
DIC_Set(cmddata, WRITE_DATA, 10, 0, (char*)data); //set data length 10, offset 10
```

```
memcpy(data,"x02\x08\x11\x12\x13\x14\x15\x16\x17\x18",10);
DIC_Set(cmddata, WRITE_DATA, 10, 10, (char*)data);//set file offset 10
errcode = DIC_Command(hic, WRITE_FILE, cmddata);//write to the current file
```

Encryption – let’s briefly take a look the method for API encryption and C51 encryption. API encryption is showed as the following program. It sends the plain text into the card and set the key ID.

```
DIC_Set(cmddata,DES_KEYFILEID,0,0x6F01,NULL);//set key file ID
DIC_Set(cmddata,DES_KEYINDEX,0,0x01,NULL);//set key ID
//set plain text
DIC_Set(cmddata,DES_DATA,BY_ARRAY | dataLength,0,(char*)DesData);
errcode = DIC_Command(hic, DESENC, cmddata);//encryption
//get encrypted text
wRet = DIC_Get(cmddata, AFTER_ENCDEC_DATA, BY_ARRAY, (char*)outData);
```

C51 Encryption:

C51 encryption is similar to the API encryption. It needs to call ‘des_enc’.

```
des_enc(fileID, keyID, 8, data);
```

Decryption, the decryption process is similar with encryption. The following code shows on how to use API and C51 to encrypt data:

```
DIC_Set(cmddata,DES_KEYFILEID,0,0x6F01,NULL);//set file ID
DIC_Set(cmddata,DES_KEYINDEX,0,0x01,NULL);//set key ID
DIC_Set(cmddata,DES_DATA,BY_ARRAY| dataLength,0,outData);
//set encrypted file
errcode = DIC_Command(hic, DESDEC, cmddata);//decryption
//set plain text
wRet = DIC_Get(cmddata, AFTER_ENCDEC_DATA, BY_ARRAY, (char*)DesData);
```

C51 decryption needs to call ‘desDec’ function as showed below:

```
des_dec(fileID, keyID, 8, data);
```

5.3.2 RSA encryption and decryption

When performing RSA encryption and decryption, the RSA key pairs need to be created firstly. Key creation could use ‘DIC_Command’ API and transfer a ‘RSAGENKEY’ macro and filling in the ‘RSA_GEN_KEY’ structure. Before performing this operation, please make sure the ROCKEY6 Smart super password is verified. If the key pairs are created dynamically at client side, then the executable files created by C51 need to call ‘rsaGenKey’. The prototype of ‘rsaGenKey’ is rsaGenKey(pubID, size, priID) : callAddress;

PubID refers to the public key file ID

PriID refers to the private key file ID

Size refers to key size (512 or 1024)

If the card doesn’t store the private key file, then creates it. The created private key file will be

inside the card; and it can be only used for computation within the card. Below are the steps for RSA encryption and decryption:

Creating key pairs – when the key pairs are creating, a public file ID, private file ID and the length of the key size need to be set (such as 1024 and 512).

API creates key pairs:

```
DICST_Rsa_GenKey keyData;//creating a struct for the key pairs
keyData.pubKeyID=0x1002//set public file ID
keyData.pubKeySize=0x1004;//set the length of the key is 1024
keyData.pivKeyID=1024;//set private key file ID
Ret = DIC_Command(Hic, RSAGENKEY, keyData);//creating key pairs
```

Creating the key pairs in C51:

```
Rsa_gen_key(0x1002,1024,0x1004);
```

API encryption and decryption

//RSA encryption

```
RSA_KEYFILEID, BY_VALUE, 0x7f01, NULL); //public key encryption
```

```
DIC_Set(cmddata, RSA_DATA, BY_ARRAY | 128, 0, dataSrc);
```

```
errcode = DIC_Command(hic, RSAENC, cmddata);
```

//RSA decryption

```
DIC_Set(cmddata, RSA_KEYFILEID, BY_VALUE, 0x7f02, NULL);
```

```
DIC_Set(cmddata, RSA_DATA, BY_ARRAY | 128, 0, EncData);
```

```
errcode = DIC_Command(hic, RSADEC, cmddata);
```

C51 encryption and decryption

//encryption

```
byte data[128];
```

...

```
rsa_enc(0x1002,128,data);
```

//decryption

```
byte data[128];
```

...

```
rsa_dec(0x1004,128,data);
```

When performing RSA and DES encryption and decryption, this library has to be imported. If the program doesn't need it, then importing the library would be optional.

5.3.3 TDES Encryption and Decryption

TDES encryption and decryption are similar with the DES. The only difference is its parameter of length is 16. In other words, the second byte of the key must be 16. For the details of the TDES

encryption and decryption, please refer to 'sample14'.

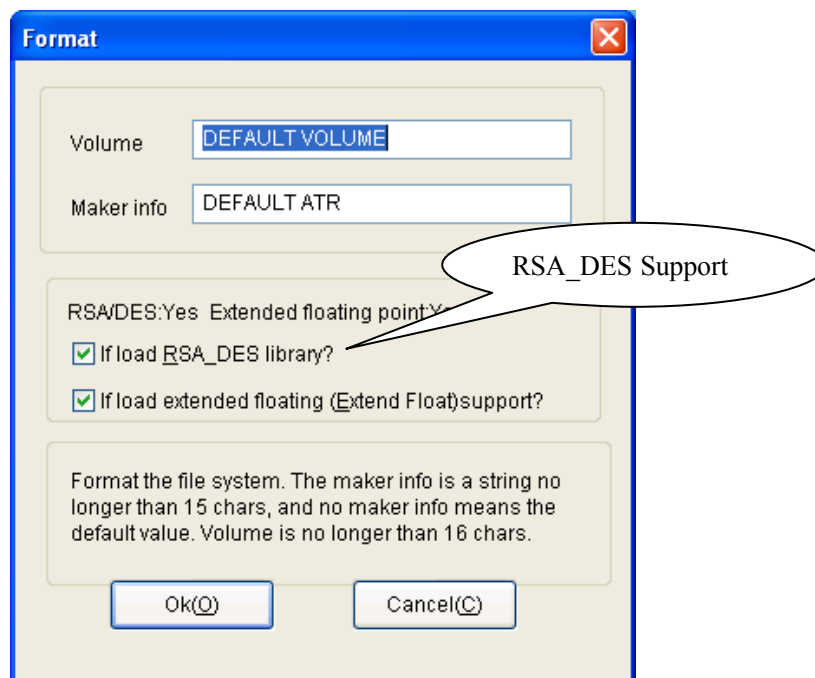


Figure 5-4

5.4 Function Expansion

The Function Expansion is used for expanding the APDU. In other words, it encapsulates and expands the current APDU commands by using the executable files. The executable file is the filtering file. By using this feature, the developers could redevelop and customize their own APDU to the dongle. Please refer to the Chapter7 of <<Advanced User Manual>> on how to use the APDU filtering.

5.5 Dongle Identification

In some cases, a user might use different kinds of dongles in the same PC. It is possible to cause problem if the program doesn't have the control function. One solution for this problem is calling 'DIC_Command' and transfers a 'GET_MANAGER_CODE' macro and a 'DICST_ManagerCode' struct (it refers to a management code, including zone code, reseller code and 2 user code). From that information, we could determine if our dongle is being used.

Another method is using management information to find connected dongles. That is, we use 'DIC_FindBuyMgrCode' function to transfer a 'DICST_ManagerCode' struct when searching dongles. The return value for 'DIC_FindByMgrCode' is the dongle handler for such manufacturer. Now, we introduce how to find and open ourselves' dongle only, when we perform the open operation.

Definition of DIC_FindByMgrCode:

```
EXTERN_C int WINAPI DIC_FindByMgrCode(DICST_ManagerCode *pMgrCode);
```

The "pMgrCode" in parameter presents the struct pointer for matching 'DICST_ManagerCode',

and searching by using the management information. Please try to use this method when the software system is released in order to avoid conflict with the other manufacturers.

```
DICST_ManagerCode m_Code;
int i=0;
int hic=0;
//struct initialization,
//here the estimated data needs to be replaced by the real data
m_Code.Zone=0x01;
m_Code.Agent=0x02;
m_Code.User1=0x03;
m_Code.User2=0x04;
int Count = DIC_FindByMgrCode ( &m_Code );
for (int i = 0; i < Count; i++ )
{
    hic = DIC_Open ( I, NULL );
    if ( hic >= 0 ) break;
}
if ( hic < 0 || hic == i )
{
    //the card is not found for operation
}
//next operation
```

5.6 Multi-thread Support

In many cases, it is necessary for many programs executing concurrently, and accessing the connected dongle independently, such as Background/Silence based server side program is designed to execute in a multi-thread development environment. A program can access the dongle via distinguished programs at the same time. This requires the dongle supporting multi-process and multi-thread programs. For those dongles that don't support multi-thread program, they have to use software to deal with this situation. There are numbers of interfaces designed for supporting multi-thread program in the API interface library. To use them, all you need to do is calling the function 'EnterMultThreadMode' before using the dongle. After calling the function, whenever the other programs open the dongle or the dongle is closed, the current program would execute correctly without any influences. When the other programs call this function and put all program processes into a process queue, all queued processes have to wait for the executable resource in order. At the same time, many processes could share the same dongle, and the dongle is in the processing sharing state. For example, when the security level for COS and the super password verification state, (There is not super password verification state in the client side) those processes can be performed in any process states.

Chapter 6 Software Release Management

6.1 Using Times Limitation

Using times limitation is commonly used for software testing. The software developers could use this function via the following steps:

- (1) Before the developers release the dongle's demo version, they have to set the using times via API.
- (2) There is required to have an executable file 'A' in the dongle. It is called every time when the software is using. In every execution time, the software will call the function 'step_counter', and deducting one from it. Please note, the program cannot only call 'step_counter' function, the function must be called with the other functions or embedded with the other functions. Otherwise, it is not hard for a cracker to find the time counter function and avoid using it.
- (3) Every time the software in the PC is called, the executable file 'A' will count itself once. Once there is only one time left for using, a pup-up window will be showed to inform the user the software is only one time to use. If the user choose ignoring the message and not to make the purchase, the software will be locked automatically after the last time using.
- (4) If the user wants to register the software, there are 2 ways to modify the time counter in the dongle. One way to do that is sending the dongle back to the manufacturer. Once the dongle is successfully passed super password verification, set the counter to zero. Another way is removing the function 'step_counter' from the program 'A'. Then using 'Remote file transfer' function to replace the executable 'A'. Now, since the 'step_counter' is removed and the executable file 'A' does not deduct any more, the software could be used without any times limitation.

6.2 Using Time Control

Using time control requires the developers to work on the internal data file to perform it.

Data file is used for (internal file can only be accessed by the internal executable file) storing time. Its accuracy is in millisecond. It has 4 bytes for saving 2 time points: 'current time' and 'end time'.

The developers need to put system time parameter into the dongle in all programs, and add the following code to the program header. The purpose of the following code is opening data file, and fetching the 'current time' and 'end time'. If the inputted time is larger than the 'end time', then the program will run, otherwise, it will prompt error message.

```
dword  pcTime;
dword  currentTime; //current time
dword  endTime;    //end time
byte   data1[8];
dword  fileId=0x1100;
get_input(&pcTime,0,2,1);//input time
open_file(&fileId, 0);    //open file
read_file(0, data1, 8);   //get the 8-byte file data
memcpy(&currentTime,data1,4);
```

```

memcpy(&endTime,data1+4,4);
if(pcTime > endTime)
{
    exit(); //time ends, returning without execution
}
if(pcTime < currentTime)
// Time error, the current inputting time is less then the previous time.
{
    exit(); //returns without execution
}

```

In addition, it has to have a program working as a ‘start program’ and ‘end program’. This program would play an important role. The ‘start program’ is used for: calculating the end time, replacing the current time with input time and starting the system clock. It is usually the first program executed in the dongle. The part of the its code is showed below:

```

dword  pcTime;
dword  currentTime; //current time
dword  endTime;     //end time
char data1[8];
dword fileId=0x1100;
get_input(&pcTime,0,2,1);//input time

open_file(&fileId, 0); //open file
read_file(0, data1, 8); //get the 8-byte file data
memcpy(&currentTime,data1,4);
memcpy(&endTime,data1+4,4);
if(endTime == 0) //initialization, only executes once
{
    endTime = pcTime+86400000; //a day is 86400000 millisecond
    currentTime = pcTime;
}
if(pcTime < currentTime)
//time error, the current inputting time is less than the last time.
{
    exit(); //return without execution
}
currentTime = pcTime;
memcpy(data1,&currentTime, 4);
memcpy(data1+4,&endTime, 4);
write_file(0, data1, 8);
start_timer(); //starting the timer

```

The ‘end program’ is used for: ending the system clock and updating the current time with the execution time + current time.

```

dword  pcTime;
dword  runTimer;
dword  currentTime; //current time
dword  endTime;     //end time
char data1[8];
dword fileId=0x1100;
get_timer(&runTimer);
get_input(&pcTime,0,2,1);//input time

open_file(&fileId, 0); //open file
read_file(0, data1, 8); //get the 8-byte data
memcpy(&currentTime,data1,4);
memcpy(&endTime,data1+4,4);
currentTime = currentTime + runTimer;
if(pcTime > currentTime)
{
    currentTime = pcTime;
}
currentTime = pcTime;
memcpy(data1,&currentTime, 4);
memcpy(data1+4,&endTime, 4);

write_file(0, data1, 8);
exit();

```

After the users purchased the software, they could use the function ‘Remote File Transfer’. Once this function is used for replacing the executable function ‘A’, the dongle could be used without any limitation.

6.3 Function Limitation

Function limited software refers to the demo software that some of its functions are not available, especially some of critical features; those missing functions or features are only offered at its purchase version. If there are two function groups for software A and B, then group A is function limited demo version, and group B is the full functional versions.

6.3.1 Using Dongle Only

Group A doesn’t have the function for calling the dongle internal algorithm. In contrast, group B has it. Group A functions correlate group A1 algorithms in the dongle; group B functions correlate group B1 algorithms in the dongle. If a user doesn’t use the full functional version of the software, then some of the functions in the group B are not available. On the other hand, if a dongle is purchased, then it is full functional version.

In this way, a developer could distribute the software freely. However, only those users made their purchase could get full functional versions.

6.3.2 Using Remote File Transfer

‘Remote File Transfer’ could be also used for the function limited demo version software. When a user purchases a dongle, the group B1 algorithms could be downloaded into the dongle by using the ‘Remote File Transfer’ function.

6.3.3 Using Remote Update Tag

‘Remote Update Tag’ could be also used for the function limited demo version software. It is similar with “Using Remote File Transfer”. The group A1 algorithms and the group B1 algorithms in the dongle are separately correlated in the group A functions and the group B functions accordingly. However, those two group algorithms are all existed in the dongle. The ‘Remote Update Tag’ is an indicator for distinguishing whether the dongle is demo version or not. That is, the last bit of the tag is exactly the position to show this. Calling the ‘getRemoteTag’ function at the beginning of the group B1 algorithms, then check if the last bit is ‘1’ or ‘0’. If it is ‘0’, ends the function without execution. If a dongle is purchased, then it has to be remote updated at the first time using it. The dongle developers will set the last bit of the tag to ‘1’. This would trigger the group B1 algorithms executing normally. Please note that the first bit of the ‘Remote Update Tag’ should be also set to ‘1’. It is correlated with the dongle itself.

Assume there are a group functions named ‘A’, and it is not available for anybody without the proper access authorities. If functions of the group A correlate with the group A1 algorithms in the dongle, then the dongle user must pass the password verification to use this function. The password verification process is carried inside the dongle. If the password is incorrect, the program will not be executed.

For the password, the PC is only an entry path. The password is actually stored and verified in the dongle rather than in the PC. If a password is 8 bytes long, those 8 bytes data will be added in the initial position of every input parameter. All the other data will be listed after those 8 bytes. The following code is an example for the password verification:

```
dword inputKey[2];
get_input(inputKey,0,2,2);//get input password

//you could also use ‘memcmp’ for comparison
if(inputKey[0] != 0x12345678)//0x12345678 is the assumed password
{
    goto end;
}
if(inputKey[1] != 0x12345678) //0x12345678 is the assumed password
{
    end:
}
```

```

        exit();
    }
    //once verified the password, a developer could write the functions here.

```

If the user is allowed to modify the password, then the password has to be saved in a data file (must be an internal data file). And save the password back to the file once it is modified. It is also possible for a data file used by many executing program. Let's assume the password is 8 bytes long. The first 2 bytes are the functional signs; the following is 8 bytes old password and 8 bytes new password. The total length is 18 bytes. Password modification is an independent executable program.

```

word option;
dword inputKey[4]; //4-11bytes are the old password, 12-19 bytes are the new password
dword realKey[2]; //get the real password from the card
dword fileId=0x1100;
//get input
get_input(&option,0,1,1);
get_input(inputKey,0,2,4);

if(option == 1)
{
    open_file(&fileId , 0); //assume the password is saved in data file with the ID = 0x1100
    read_file(0,(byte*)&readKey,8); //get the real password
    if(inputKey[0] == realKey[0]){
        if(inputKey[1] == realKey[1]){
            write_file(0,(byte*)&inputKey+2,8);
            //write the new password into the data file
        }
    }
}
}

```

6.4 Multi-level Authorization Control

Assume there are two functions groups A and B. the user A0 could use both of them, and user B0 could only use function of the group B. Additional, the group A functions correlate with the group A1 executable programs in the dongle, the group B correlate with the group B1 executable programs in the dongle. If assigning the same password to the executable program of the group A1 and B1, then the user A0 and B0 can use the both function groups. However, if the group A1 and B1 assigned different password, then the user A0 has to inconveniently remember the both passwords.

The ideal solution is: both the user A0 and the user B0 use the same password. Namely, the group A1 executable programs could be used under the same password. The group B1 executable programs could be used under the 2 passwords. The executable programs group A1 is using the same password verification process like above. The password verification process for the

executable program group B1 is also simple. It actually checks two passwords to see if they are identical. Below is an example with 8-byte assumed password:

```
dword inputKey[2];
get_input(inputKey,0,2,2);
if(inputKey[0] == 0xA1234567){ // check the password of A0
    if (inputKey[1] == 0xA1234567){
        goto right;
    }
}
if(inputKey[0] == 0xB1234567){ //check the password of B0
    if (inputKey[1] == 0xB1234567){
        goto right;
    }
}
exit();
right:    //password is correct
//the other code goes here
```


Chapter7 Remote Update

‘Remote File Transfer’ is used for the developers to conveniently update their software system. Assume a developer has sold lots of 1st version of their software; the users who used the 1st version software will make the purchase and update to the 2nd version of the software.

In this update example, assume the algorithm used in 2nd dongle is different with the algorithm in the 1st dongle. Then the developers don’t need to replace the old dongle. All the developer needs to do is to send the new algorithm to the user by using the ‘Remote File Transfer’ function. From the users side, the users need to download the new algorithm into the dongle via the ‘Create Plain Text’ function.

Now, we are going to introduce some basic methods for remote upgrading, and then we will describe them in details by showing some examples.

7.1 Multi-module Management

‘Multi-module Management’ refers to writing a numbers of code modules in a same dongle. The PC programs contain the correlated modules too. In most cases, the users only purchase some of the modules; the rest modules are purchased only if the users need them. In this section, we will show how the developers to manage the sold dongle, and adding the dongle internal modules without any modifications.

Actually, “multi-modules” means the developers could divide software into many parts. Some of them are used for the software demo version; the rest of them may be used in the full functional purchase version. According to this, if we group all programs in the full functional version software and combine with corresponded C51 programs to a module, then we can use the following function to perform the module management. At the examples attached with this Tutorial, there is an example looks like the below. The example shows the most of functionalities of the Rocky6 Smart dongle. And it can perform the module management by using the remote management tools provided by the ROCKEY6 Smart dongle.

The screenshot shows a software window titled "tutorial" with a blue title bar and a red close button. The window contains several sections for cryptographic and file operations:

- Floating computation-Cubic**: Includes input and output fields (both set to 0) and a "DOUBLE" button.
- Create key pair**: Includes fields for "Public key:" and "Private key:", and a "Create RSA Key" button.
- Create file**: Includes fields for "File ID:", "File type:", and "Size:" (set to 0), and a "CreateFile" button.
- Write file**: Includes fields for "File ID:", "Offset:" (set to 0), and "Length:" (set to 0), and a "WriteFile" button.
- Read file**: Includes fields for "File ID:", "Offset:" (set to 0), and "Length:" (set to 0), and a "ReadFile" button.
- DES encryption/decryption**: Includes fields for "File" and "Key ID:", an "Encrypt" checkbox, and a "Run Des" button.
- RSA encryption/decryption**: Includes fields for "Public key ID:" and "Private key ID:", an "Encrypt" checkbox, and a "Run RSA" button.
- Data**: A text area displaying a list of hexadecimal values (e.g., 00000000, 0000000C, 00000018, 00000024, 00000030, 0000003C) followed by a series of dots. The text area has a vertical scrollbar.

At the bottom of the window are "Cancel" and "Ok" buttons.

Figure 7-1

7.2 Using Remote Update Tag

This method is designed to handle the situation that the total number of the modules in the dongle is less than 32. A 'Remote Update Tag' is a 4-byte binary value. The 1st bit on the left (the Most Significant Bit) is used for expressing if the remote update is related with the hardware. The rest of 31-bit is available to use. There is 1-bit out of the 31-bit used to represent the initiation state for a module.

For example, if the last bit (the Less Significant Bit) is used for representing the initiation state for a module 1; '0' means no initiated; '1' means initiated. After the module 1 corresponded C51 program in the dongle accepting the input data, use 'getRemoteTag' function to get remote update tag. And then check if the last bit is '1'. If it is '0', then return it without execution; otherwise, execute it.

Once a user purchases a dongle and needs to buy a new module, he/she needs to send the dongle remote update tag to the developer. The developer will set the value of the module to '1' after

receiving the user's tag; then generate a new remote update password and send back to the user. The user could use the new module after applying the received update file. In the example of this Tutorial, every example's entry point contains the code similar to the following:

```
Ret = get_remote_tag(&remoteTag);
remoteTag&=0x01;
if(remoteTag==0)
{
    set_response(1,&Ret);
    exit();
}
```

This piece of code is used for getting the remote update tag from the program, and getting its initiation bit position from the 31-bit to determine if the program will continuously execute. If there is not proper initiation bit position found from the 31-bit, then the program will pop up an error message and quit without execution. So, it is the way to perform the module management.

7.3 Remote Module Management

If a developer write the numbers of module greater than 32 in the dongle, then the 'Remote Module Management' function is used to handle this situation. The 'Remote Module Management' could be set by using the dongle IDE configuration and remote user settings.

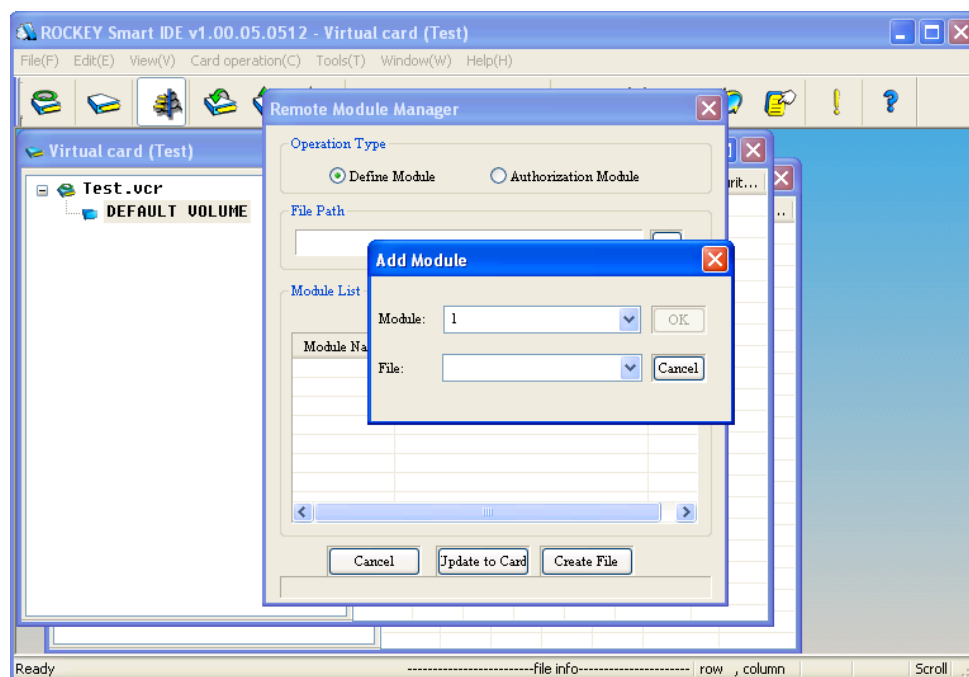


Figure 7-2

To config the local ROCKEY6 Smart dongle for its module management, we need to add the dongle files into the list, set the open/close states, and update the device.

To remotely manage modules, the developers need to generate a module definition file. The users will create their own module request file according to the module definition file. And then the users will send the module request file back to the developers. Once the developers receive those files, they will generate the module authorization file and send back to the users. When the users receive and apply those authorization files, the remote module update process is accomplished. For the details of module management, please also refer to <<User Manual>> and 'client side tools'.