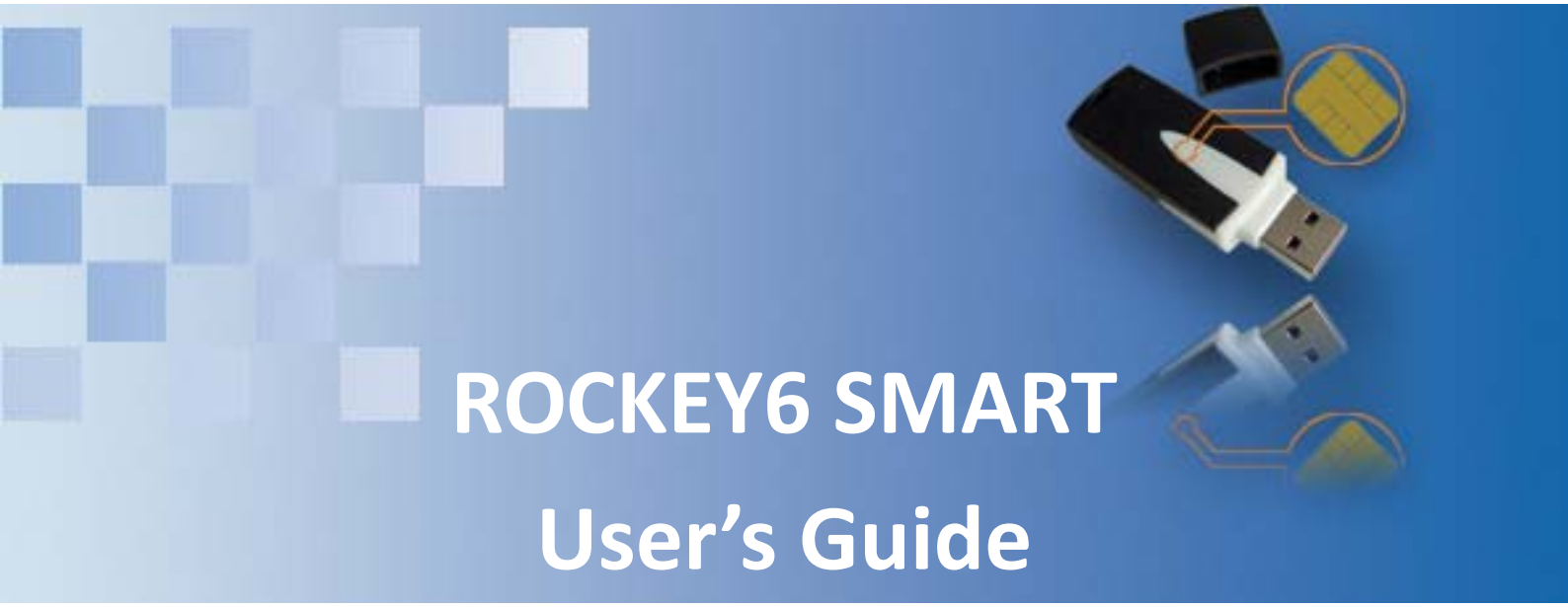
FEITIAN



ROCKEY6 SMART
User's Guide

V2.0

Feitian Technologies Co., Ltd.

Website: www.FTsafe.com

Revision History:

Date	Revision	Description
Sep. 2008	V1.0	Release of the first version
Dec. 2009	V2.0	Release of the second version

Software Developer's Agreement

All Products of Feitian Technologies Co., Ltd. (Feitian) including, but not limited to, evaluation copies, diskettes, CD-ROMs, hardware and documentation, and all future orders, are subject to the terms of this Agreement. If you do not agree with the terms herein, please return the evaluation package to us, postage and insurance prepaid, within seven days of their receipt, and we will reimburse you the cost of the Product, less freight and reasonable handling charges.

1. Allowable Use – You may merge and link the Software with other programs for the sole purpose of protecting those programs in accordance with the usage described in the Developer's Guide. You may make archival copies of the Software.

2. Prohibited Use – The Software or hardware or any other part of the Product may not be copied, reengineered, disassembled, decompiled, revised, enhanced or otherwise modified, except as specifically allowed in item 1. You may not reverse engineer the Software or any part of the product or attempt to discover the Software's source code. You may not use the magnetic or optical media included with the Product for the purposes of transferring or storing data that was not either an original part of the Product, or a Feitian provided enhancement or upgrade to the Product.

3. Warranty – Feitian warrants that the hardware and Software storage media are substantially free from significant defects of workmanship or materials for a time period of twelve (12) months from the date of delivery of the Product to you.

4. Breach of Warranty – In the event of breach of this warranty, Feitian's sole obligation is to replace or repair, at the discretion of Feitian, any Product free of charge. Any replaced Product becomes the property of Feitian.

Warranty claims must be made in writing to Feitian during the warranty period and within fourteen (14) days after the observation of the defect. All warranty claims must be accompanied by evidence of the defect that is deemed satisfactory by Feitian. Any Products that you return to Feitian, or a Feitian authorized distributor, must be sent with freight and insurance prepaid.

EXCEPT AS STATED ABOVE, THERE IS NO OTHER WARRANTY OR REPRESENTATION OF THE PRODUCT, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5. Limitation of Feitian's Liability – Feitian's entire liability to you or any other party for any cause whatsoever, whether in contract or in tort, including negligence, shall not exceed the price you paid for the unit of the Product that caused the damages or are the subject of, or indirectly related to the cause of action. In no event shall Feitian be liable for any damages caused by your failure to meet your obligations, nor for any loss of data, profit or savings, or any other consequential and incidental damages, even if Feitian has been advised of the possibility of damages, or for any claim by you based on any third-party claim.

6. Termination – This Agreement shall terminate if you fail to comply with the terms herein. Items 2, 3, 4 and 5 shall survive any termination of this Agreement.

PREFACE

ROCKEY6 SMART dongle is a newly developed powerful security product from FEITIAN Technologies. It is a smart card kernel based USB device that combines software protection dongle and smart card technologies in a very small form factor. It supports multiple applications including software protection, personal identification and authentication, data security and protected electronic transactions. This manual will guide you on how to use ROCKEY6 SMART to protect and encrypt your products.

PART1 Basics

All ROCKEY6 SMART specific features, functions and basic concepts for software protection will be introduced in this part, providing you with a general overview of this product.

PART 2 Applications

After completing this part you will know how to use Keil u Vision2 to compile the kernel of your encrypted application and choose the corresponding API function. You will also learn the development procedure with ROCKEY6 SMART.

PART 3 Advanced Functions

This part will introduce RSA, DES encryption algorithms and how to work with APDU and COS (Card Operation System). After completing this chapter you will not only upgrade to a further level of using ROCKEY6 SMART to protect your application but you will also master the advanced encryption management features.

In addition, Appendix A describes all terms involved in this manual.

The following is some points you may need to know:

1) The initial super password of ROCKEY6 SMART demo dongle is 8 bytes: FF FF FF FF FF FF FF FF. The retry times limit is 15, i.e. the smartcard will be locked after 15 times continuous unsuccessful verifications. If the dongle is locked, it must be returned to the manufacturer to be repaired. Super password needs to be verified before the advanced operation like generating or deleting executable files or internal files, formatting dongle, generating remote update file etc. Please keep the super password safe. Do not include it to the encrypted program and never show it to others. If change the super password to eight 0x00 (i.e. 00 00 00 00 00 00 00 00), the smartcard will be locked. In this case, no files can be imported to the dongle any more. Only the executable files can be called. Using initial super password in development is recommended. This can prevent the smartcard being locked unexpectedly. Once the smartcard is locked, there will be no way to unlock it, even the manufacturer cannot fix the password or unlock the dongle. This design can guarantee maximum security performance.

2) When ROCKEY6 SMART is inserted, the flash disk icon will be displayed in the task bar. User can unplugged the dongle directly when finish using.

3) When ROCKEY6 SMART dongle is attached to the computer, if the light of the dongle blinks, this indicates a connection error. Please check the operating system status, the USB port connection. In Windows98 system,

please check is the driver installed properly. Switching to a different USB port or test with a flask disk may help to find the problem.

4) The memory space of ROCKEY6 SMART dongle is about 64 KB. If FLOAT library and RSA_DES library are loaded, there will be about 28 KB left available. The space is located at EEPROM with 500,000 rewritten times. The rest of memory is located at FLASH with 20,000 rewritten times. Reading times is not limited. RSA_DES library costs about 14 KB space and FLOAT library costs about 21 KB space.

5) There are three ways to load executable programs to the dongle:

Use "Download to Flash Memory" button in KEIL

Use ROCKEY6 SMART IDE tool to import BIN file generated by hexbin.exe

Use ROCKEY6 SMART API to import the program

6) An advice for programming, one process should only call open and close dongle once. When start the process, open the dongle, and when close the process, close the dongle. For multi-process and multi-entry operation, please add mutex mechanism to the design.

7) C51 program has three ways to accept external inputs. When acquire BYTE input and store it in integer variable, please alternate the order of the input bytes. The output of C51 program is in BYTE order, please also alternate the order of integer variables (refer to sample 11). For double float variable (DOUBLE type, 8 bytes), the order need not to be changed.

8) If do not want the files to be listed with ROCKEY6 SMART IDE, there are two ways to prevent this:

Set executable file's attribute as internal executable. File will only be listed after verified super password.

Using APDU filter, software vendor can define the APDU command of listing files (refer to section 12.4).

9) Management code is composed of four parts (Zone, Distributor, User1 and User2). This information is preset during the manufacturing. User cannot change them. Management code is used for sales management. Different software vendors have different management code. User cannot buy dongles from other vendor which has the same management code. It is the important information to identify dongles (refer to sample18).

10) Remote update cipher text file is related to the user code and update password (if password is set). This means the same plain text file can generate different cipher text with different user code or remote update password (refer to section 5.2 Secure File Transfer).

11) Virtual card cannot stimulate all the functions of real card, for example, RSA_DSE function and a C51 program calling other C51 programs. Developing applications with real card is recommended.

12) When called by external API, executable programs are identified by their file name, not the file ID. Using same file name and file ID is recommended. For example, file ID: 0x1008 will have the file name "1008". File name cannot be overlapped.

13) One note for using counter of ROCKEY6 SMART, if the specific number of times is set, data cannot be processed once the counter decline to 0. It is supposed to start operation while number is greater than 0, or the counter of ROCKEY6 SMART will be un-reused for the user, and must be returned to the manufacturer to be initialized for the second time.

Contents

PART 1 Basics

All ROCKEY6 SMART special features, functions and basic concepts for software protection will be introduced in this chapter. You will obtain a general overview on ROCKEY6 SMART.

Chapter 1, ROCKEY6 SMART Software Protection Solution

In this chapter you will learn the specifications, the functions of ROCKEY6 SMART and theory of protecting your application. After completing this chapter, you will obtain a general overview of ROCKEY6 SMART.

Chapter 2, ROCKEY6 SMART Installation and Removal

Before using ROCKEY6 SMART, you should first install the ROCKEY6 SMART Developer Kit. This chapter will explain the procedure to both install and remove the ROCKEY6 SMART Developer Kit.

Chapter 3, Integrated Development Environment (IDE)

In this chapter you will learn the following,

1) How to manage the card. It includes managing internal files, formatting cards and setting the password;

2) Manage to download executable files;

3) How to implement the remote management.

Chapter 4, Card Operation System (COS)

ROCKEY6 SMART is based on IC card technology. COS regards the electrical chips as a small virtual computer. ROCKEY6 SMART has its own CPU and file system.

Chapter 5, Remote Control

Remote control involves three parts, remote update control, secure file transportation and remote model control. You can update your software and the encryption algorithm, and also manage the user model (shut or open the corresponding model) securely.

Chapter 6, Manufacturing Management

You will learn how to program ROCKEY6 SMART in batch in this chapter.

Chapter 1. ROCKEY6 SMART Software Protection Solution

1.1 ROCKEY6 SMART Introduction

The ROCKEY6 SMART dongle is another powerful product from FEITIAN Technology Co., Ltd. It is the result of research and insight gained from the development of the ROCKEY5 dongle, ePass1000, ePass2000 and Feitian's experience in driverless technology. The ROCKEY6 SMART dongle is a USB device which is based on IC card technology utilizing encryption technology. It can be utilized in many areas such as software protection, authentication, and electronic business and information security.

ROCKEY6 SMART Features:

✓ Security at Hardware Level

The ROCKEY6 SMART is based on smart card technology. Smart cards are widely used in the banking and financial sectors where security is of paramount importance. Smart card hardware is engineered to prevent reverse engineering and specialized analysis, which used by hackers and others who seek to crack security schemes. Smart cards integrate modules such as CPU, RAM, EEPROM and FLASH and are essentially a mini-computer on a chip, providing a powerful and versatile platform for developing complex and powerful security programs.

✓ Hardware Compatibility

ROCKEY6 SMART consists of a USB card reader and a specially programmed smart card.

✓ Security at Software Level

The target program's kernel algorithms and data may be "migrated" and executed inside the smart card's protected environment. The migrated code operates under the management of the smart card operating system and forms a miniature-computing environment that runs in parallel to the main computer. The ROCKEY6 SMART environment and the main computing environment exchange data through the USB port. The probability of a successful hack of this system can be reduced to near zero if a reliable solution plan and certain sophisticated algorithms are correctly adopted.

✓ Software Compatibility

ROCKEY6 SMART fully supports the ISO7816 smart card standard as well as specific extensions to the standard. This system will be familiar both to users of software protection dongles as well as those familiar with smart card technologies.

✓ Flexibility

Many advanced administrative functions (such as Remote Update Management) have been incorporated for software developers. All security requirements are accomplished through a set of basic functions provided with the ROCKEY6 SMART system. Developers may build their own complex protection plans based on the ROCKEY6 SMART function set.

1.2 How to Protect Software with ROCKEY6 SMART

By using ROCKEY6 SMART, you can migrate a portion of the target application into the ROCKEY6 SMART memory, with the rest residing on the computer. The applications which are migrated into the ROCKEY6 SMART are called "external programs". The external program will not be loaded into the main computer memory during execution. External programs are stored in Electrically Erasable Programmable Read Only Memory (EEPROM). The main program can call the external programs separately – it cannot call them simultaneously. External programs can also call one another. The target application cannot function properly without the dongle attached.

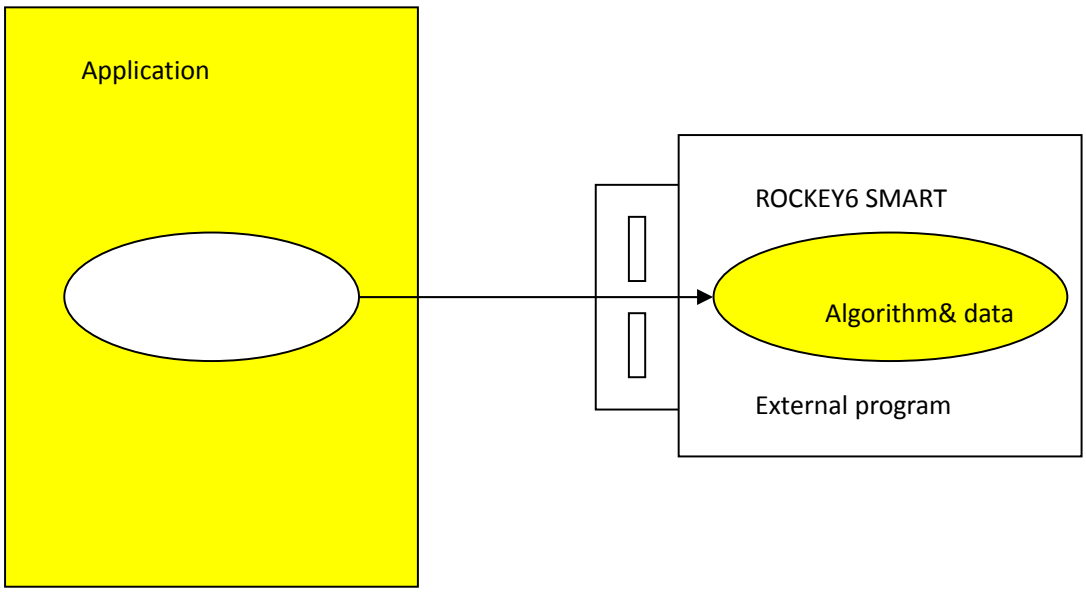


Figure 1-1 ROCKEY6 SMART Software Protection Diagram

1.3 How to Use ROCKEY6 SMART Protected Software

The ROCKEY6 SMART dongle is a driverless device. You can connect the dongle directly to the computer without having to install any USB device drivers.

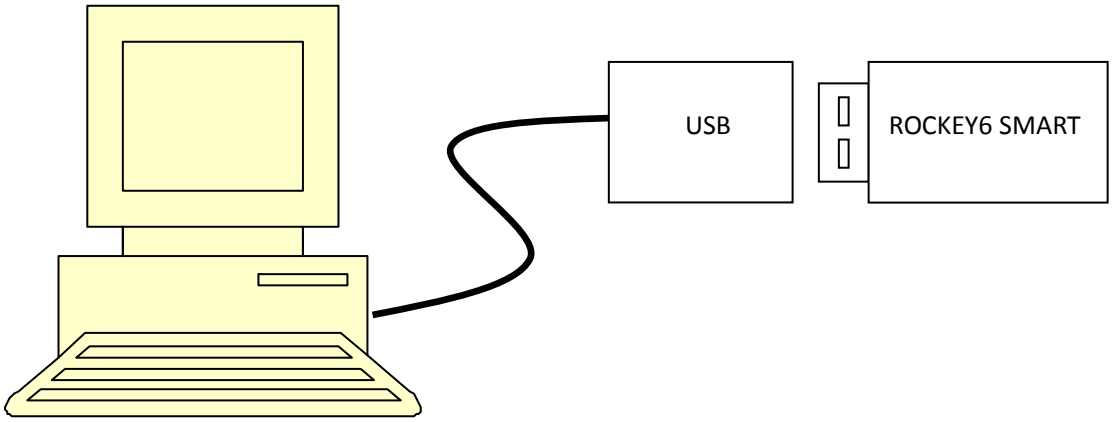


Figure 1-2 Using Software With ROCKEY6 SMART

1. When the main program execution reaches an external program, the program call command is sent to ROCKEY6 SMART along with the required parameters.

2. ROCKEY6 SMART responds to the command of the main program, executes the related external program and returns the result back to the main program. ROCKEY6 SMART remains in waiting status until the next command.

3. You will obtain the results from the dongle as if it were executed all on the computer; however the procedure runs in the dongle and never loads into the computer.

Chapter 2. ROCKEY6 SMART Installation & Removal

On the ROCKEY6 SMART installation CD, you will find the directory ROCKEY6 SMART SDK. Double-click "setup.exe" to start your installation.

2.1 Installation

✓ Step1: Welcome Window

After double-clicking "setup.exe", the following window is shown:

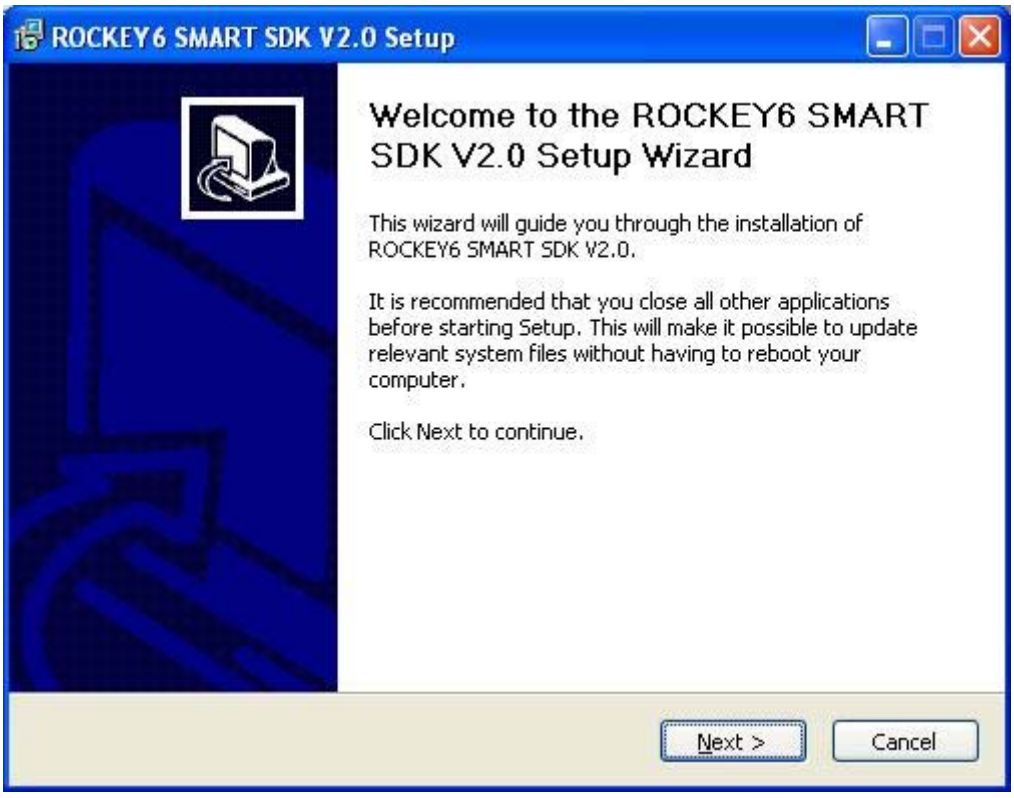


Figure 2-1 Welcome Window

✓ Step 2: Agreement

After you choose [I Agree] the agreement, click [Next] to continue:

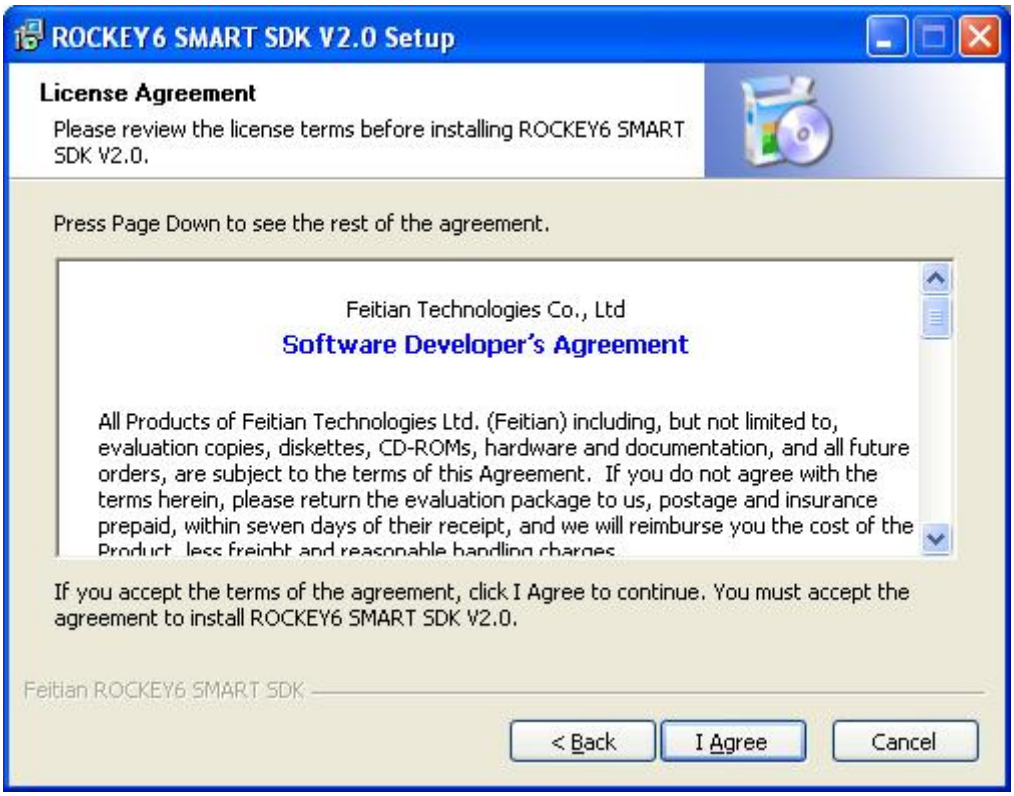


Figure 2-2 Agreement

Step 3: Select Installation Model and Components

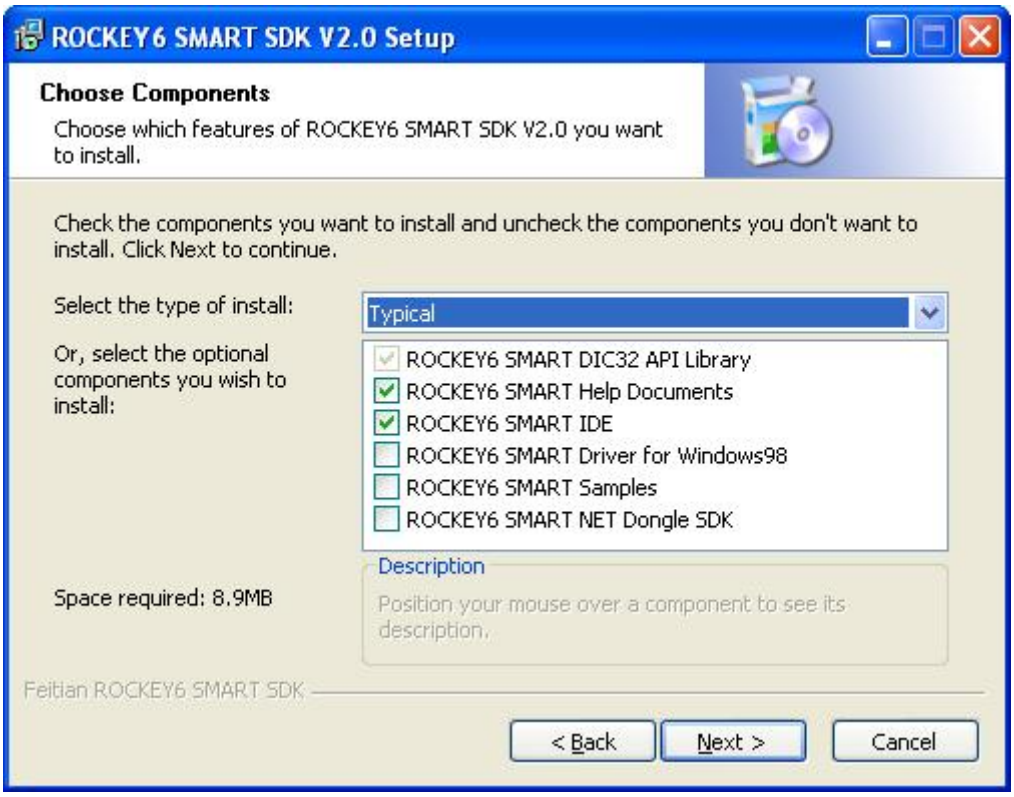


Figure 2-3 Installation Type Selection

- If "Typical" option is chosen, the program will automatically install the ROCKEY6 SMART API library,

ROCKEY6 SMART IDE and help documentation for you.

- If "Compact" option is chosen, the program will only install the ROCKEY6 SMART API library and developer manual.

- If "Custom" option is chosen, the user can select the components needed to be installed.

Step 4: Install Location

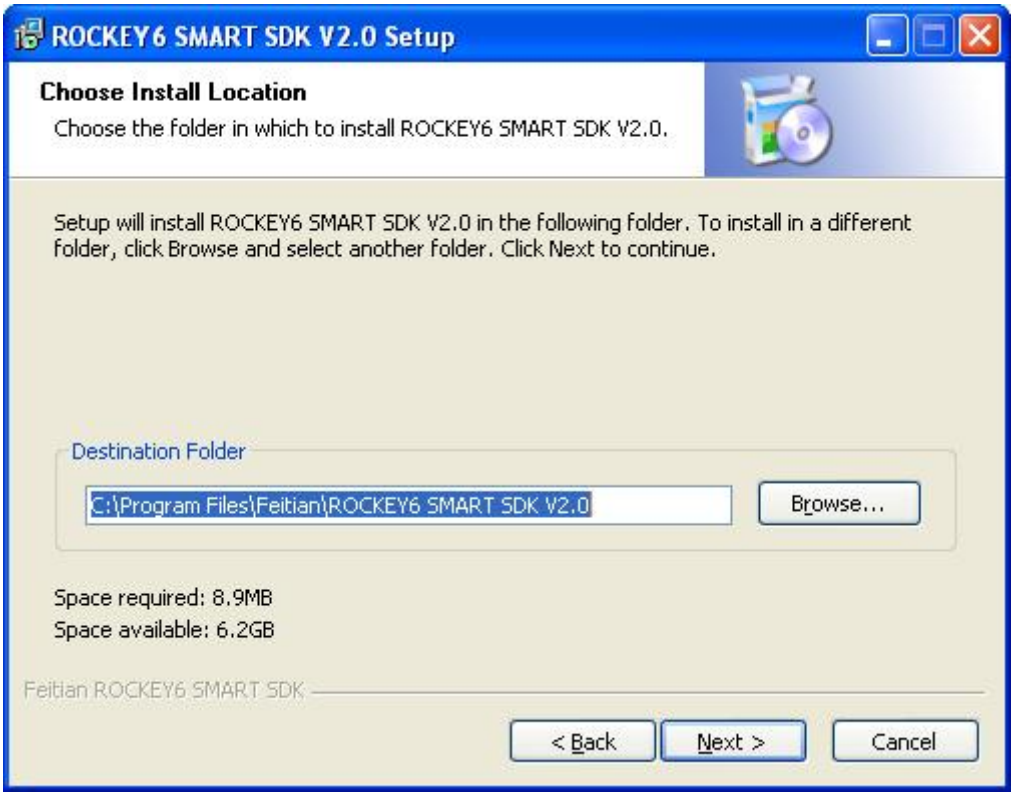


Figure 2-4 Path Selection For Installation

✓ Step 5: Installation is complete

Once completing the installation, click "Finish" button to quit.

Figure 2-5 Complete Installation

2.2 Removal

There are two methods to remove the installation: One is from "Control Panel" —> "Add or Remove Programs" where you select "ROCKEY6 SMART" to delete it; another is from "Start" —> "Programs" —> "FEITIAN" —> "ROCKEY6 SMART SDK V2.0" -> "Uninstall ROCKEY6 SMART SDK V2.0".

2.3 Driver Installation

ROCKEY6 SMART is a USB device that works smoothly with Windows and Linux systems. If ROCKEY6 SMART is plugged into the USB port, the system will automatically detect that the drivers are not installed in the computer and initiate the driver setup wizard.

For Windows 98 systems, a Win98 installation disc is required for the driver installation. For other operating systems such as Windows Me/2000/XP, Windows 2003 server and Vista as well, the driver setup wizard will automatically start.

Chapter 3. Integrated Development Environment (IDE)

The Integrated Development Environments has the following features:

- ✓ Card management, it includes formatting, password and volume initialization and updates etc.
- ✓ Executable file management for downloading
- ✓ Remote control (for details see Chapter 5)

3.1 IDE Overview

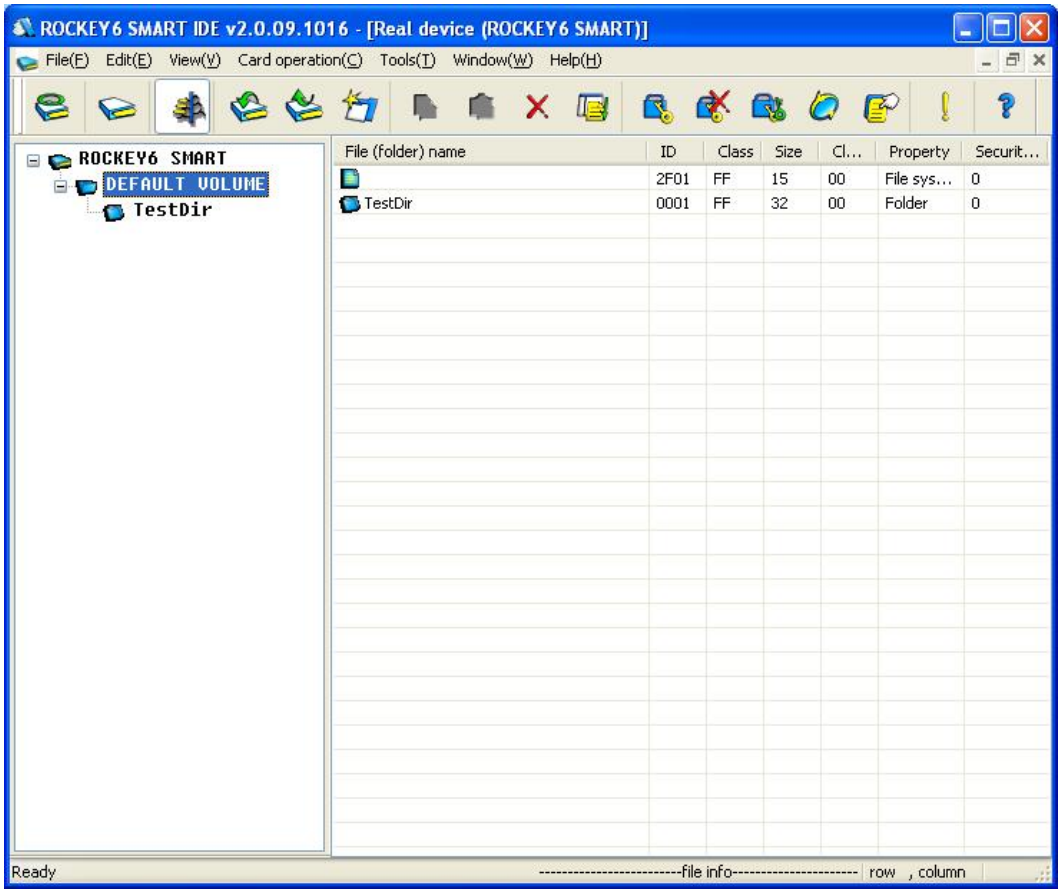


Figure 3-1 IDE

From Figure 3-1 IDE, the standard IDE includes menu bar, tools bar, project bar, edit window, compile bar and status bar. Here, we only provide an example for "Project" bar, "Tools" bar, "Detected Device" window and "Virtual Device" window.

3.1.1 Menu Bar

Menu bar encompasses all the commands to perform all IDE functions. The following table lists all items from the menu bar. Menu items are dynamically designed, meaning that they appear only when they need to. Therefore users may view different menu items according to cases. Their functions will be discussed in the main menu section.

Some menu items in our IDE example:

Table 3-1 File Menu

New C51 Project	Create a new project
Browser Real Devices	Browser Plug-in Devices
New Virtual Device	Create a new virtual device
Open Virtual Device	Open a virtual device file
Recent Files	List recently opened files
Quit	Quit program

Table 3-2 View Menu

Toolbar	Display or hide Tools bar
Status Bar	Display or hide Status bar
Device Tree	Display or hide files tree of the device

Table 3-3 Card Operation (Appears only at device window)

Refresh	Refresh current active devices
Format	Format device file system
Modify Password	Modify super password for device system
Remote Update Management	Create and test remote update password
Remote Module Management	For manufacture remote module management
Generate Cipher text file	Generate a cipher text file
Generate Plain text file	Generate a plain text file
Card Information	Display information for manufacturer, hardware and management ID
Burn Real Card	Burn the content of selected virtual device into real card
Verify the Super Password	Switch to the state for verifying the super password to display the internal files
Cancel Verification State	Cancel the state for verifying the super password to hide the internal executable files
Import File	Import disc file into device
Export to Disc	Export file to the disc
New Directory	Create a new subdirectory under the file system of the device
Run	Run the selected executable file

Table 3-4 Tools Menu

Binary Code -> Source Code	Convert binary code to arrays in C,VB source code

Table 3-5 Windows Menu (Appears at sub-window active state)

Cascade	Cascade the windows
Tile vertical	Arrange windows by column without cascade
Tile horizontal	Arrange windows by row without cascade
Close Window	Close current active windows
Close All	Close all active windows
Previous Window	Switch to previous window
Next Window	Switch to next window

Table 3-6 Help Menu

About ROCKEY6 SMART IDE	Display software information, version number, copyright, API version number and hardware information

3.1.2 Toolbar

Tools bar may be displayed differently each time according to distinguished active window types.

Table 3-7 Common Button Across Different Statuses

	Open a virtual device file
	Browse connected devices
	Display software information, version number, copyright, API version information and hardware information

Table 3-8 Useful buttons in system initial state (before active window popping up)

	Create a new C51project

Table 3-9 Useful buttons at the state of editing window is active

	Save active content
	Cancel previous modification
	Cut the selected subject and save it to the clip board

	Copy the selected subject and save it to the clip board
	Paste the saved subject from clip board

Table 3-10 Useful buttons when device window is active (includes real devices and virtual devices)

	Import files from the disc into the device
	Export files from the device to the disc
	Create a new sub directory in the device file system directory
	Copy the selected file subject and put it on the clip board
	Paste the content of file subject from the clip board
	Delete file or directory. When you delete a directory, please make sure the directory is empty before its deletion.
	Display the content of the selected device file.
	Switch to the state of super password verification, in order to display the internal files.
	Cancel the state of super password verification, in order to hide internal executable files.
	Modify the password for the device file system.
	Refresh the devices.
	Display manufacturer information, hardware information and management code.
	Run the current executable file. (If it is an executable file)

3.2 Writing Files to ROCKEY6 SMART

After the software system is successfully initialized, the user can write files into the ROCKEY6 SMART. The initialization of ROCKEY6 SMART includes setting up the label, manufacturer information and super password. The user can modify the default values by simply formatting the ROCKEY6 SMART.

Figure 3-2 displays the formatting window for ROCKEY6 SMART. It is different from setting up the virtual file system. The label and manufacturer information in the virtual file system contains default values that cannot be modified.

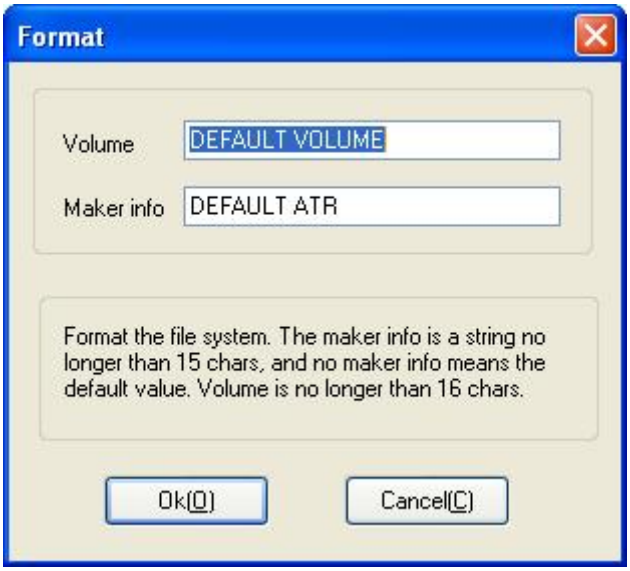


Figure 3-2 Formatting ROCKEY6 SMART

Before formatting, select the necessary modules for loading. Writing a file into the ROCKEY6 SMART is easy, similar to copying a file between folders. It can be done by using the following methods:

1) "Ctrl + C" (Copy) then "Ctrl + V" (Paste) or right clicking the mouse and selecting "Copy" and "Paste". This method is commonly used for adding a file to the dongle or copying an executable file from one virtual card to another.

2) The difference between directories in a PC and ROCKEY6 SMART is that the executable file in the real card can only be deleted and not be read. Some manipulations require super password verification like deleting files, however some other manipulations like copying one file from one ROCKEY6 SMART to another, or viewing a text file and performing an executable file do not.

Chapter 4. Card Operation System

4.1 Modules

ROCKEY6 SMART uses an IC card, integrated circuit chip, as the core for its algorithmic execution and storage medium. The design of this card's operating system enables the entire device to work as a virtual minicomputer which has its own CPU and file system.

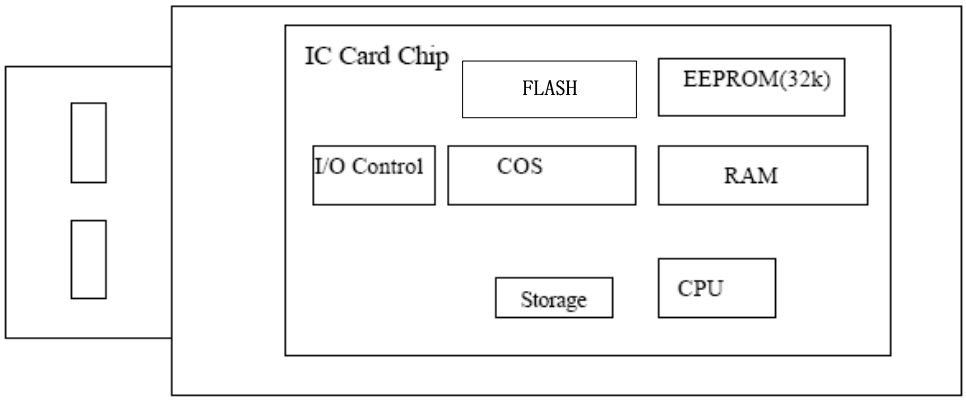


Table 4-1 IC Card Structure

The EEPROM works similar to a computer's hard drive which can be used for storing programs and data. All files in the ROCKEY6 SMART are stored in EEPROM, with these files being only read and written by its manufacturer or its integrator but not End Users. End Users are unauthorized to access data in the EEPROM. There is only one file system in the EEPROM with every operation being performed in terms of file-operations (open, close, read, write, delete etc.). The EEPROM could be treated as a logical driver in the PC with the entire file system marked to a volume (root directory), just likes a logical driver on Drive "C:\ ". Common operations can be performed in the logical driver, such as making directories, and writing and removing files, however, some operations require super password verification.

The RAM area is similar to a computer's memory where the application input, output and temporary variables are store.

COS (Card Operation System) controls the entire IC card system. In the process of communication between ROCKEY6 SMART and the host computer, ROCKEY6 SMART always passively receives instructions from the host computer, never visa-versa. All messages from the host computer are interpreted and executed by the COS.

4.2 Data Files and Directories

The file system contains data files, executable files and directories. In this section, we explain the data files and directories.

Data files and directories are distinguished by a 2-byte ID and located in the same name space. In the root directory, some IDs are held by the system default directories and files. These ID's are 0000 (reserved by system), 2F01 (ATR files - includes manufacturer information), 3F00 (volume), and 3FFF (current directory). Any one of above four ID's holds a complete name space. Except for the above mentioned ID's, all other ID's can be used freely.

Volume and manufacturer information can only be set and modified when the ROCKEY6 SMART is formatted. The content of manufacturer information is 15 bytes long and written by the manufacturer. If the manufacturer chooses not to input any data, then COS will automatically fill them with default values. The volume is 16 bytes long and also set by the manufacturer.

4.3 Executable Files

All executable files are recognized by their ID's and file names. The ID's for executable files, directories and data files use the identical name space. We suggest users allocate ID's starting from 0x1000 when the data files and directories are created. This will avoid ID conflicts.

Naming an executable file (file name is 16 bytes long) is a little complex. An executable file is usually selected according to its name. The following explanation covers this technical detail. For those who are not interested in this may want to jump to Section 4.4 directly.

According to ISO7816-4, we assign the following rules for naming the executable files.

File name is "xxyy", 2-byte. "x" or "y" is a hexadecimal number. The following is the range for "xx":

Table 4-1 Values of xx

xx	Description
10 - 65	User-defined value range 1
67 –7F	User-defined value range 2
D0 - EF	User-defined value range 3
F1 - FE	User-defined value range 4

The following rules are used for the value range of "yy".

1. Cannot be a 6x or 9x
2. Cannot be odd numbers
3. "66" is a reserved wildcard character. Please do not use it due to its complexity.

The rest are also reserved values for ISO7816-4. They are possibly used by executable files, but still complicated when using them.

All executable files are stored in the root directory. In contrast, data files can be stored anywhere in the directory. An error message of COS system will prompt if moving executable file into the subdirectory, and accordingly, the executable file cannot be touched by system this way. In the COS system, ONLY executable files in the root directory can be executed.

4.4 File Classes

Directory classes and file classes are symbolized by a byte (0x00 – 0xFF) named File "FCLA". It is initially defined by the manufacturer. All files with the same FCLA can be treated as files from the same product. With the restriction of FCLA, all executable files in the ROCKEY6 SMART can only read, write and create the data files with the same FCLA. The following rules are applied for FCLA:

- ✓ 0xFF represents a "null class", in which all its data files can be accessed by any type of executable files, but all executable files in this class can only access the data file with 0xFF.
- ✓ The class of the root directory is also a "null class".
- ✓ All files with the same class constitute a relatively independent subsystem. All data files of a "null class" are public data; and all executable files of "null class" are public programs.

FCLA is commonly applied for product management. If a manufacturer is developing many products, or a number of manufacturers are cooperatively working on the same product, FCLA would be good to apply. For example, if a dongle needs a set of application software, then set FCLA to "FF". All files under the same directory and subdirectory would be set to the same FCLA; and their accessibilities controlled by "security level".

File class is recorded in the system with 1-byte length called "system current file class". When "system current file class" is executing "file selection" or fetching the executable files in the dongle, it can switch according to the

following rules:

(1) If the target file is a "null class" file, then the system current file class will keep its class unchanged; otherwise they will switch to the class of the target executable file.

(2) Whenever the class switching occurs, the "system current security level" will be reset to "0".

(3) After resetting the dongle, the system current file class's default value is 0xFF (null class).

Another important usage for FCLA is filtering files; however, this topic is beyond the scope of this manual. For further information, please contact us.

4.5 File Security Level

Beside the class, every file has a "security level" for controlling the access to files with the same class. Namely, "security level" is set for the files with the same class.

In most cases, the "security level" combines 4-bits of low order byte (from least significant byte), and attribute (4-bit of high byte [from most significant byte]) into a byte. There are 16 distinguished states in total. In those states, "0" is the lowest level and "15" is highest.

COS also has its own security level. At run time, it is recorded by a byte. This security level has to be updated by the ROCKEY6 SMART internal executable file (via system fetching).

However, there is a constraint – executable files cannot grant COS security levels higher than itself. Executable files can only access (read, write or create) data files with a lower security level. When finished, they can set the security level to "0".

If you do not need this high-level management function, set all security levels to "0". Note that the permission for creating file is separate from the permission for writing to file according to the COS design. Thus, file creation can be successful when importing a file the security level of which is higher than "0", while the permission is not enough for writing to it. In this case, the file can be enumerated. But its content is null.

4.6 File Properties

There are 5 file property types: Normal, Internal, Up-to-Ignore, Directory and Executable. A directory or file can have many several properties.

- "Normal" is the default property for a file without any customized settings.
- "Executable" indicates the file is executable for COS. However, those files have to be allocated at the ROCKEY6 SMART root directory and can only be created or deleted after successful super password verification.
- "Dir" means it is a sub-directory. If this property is set, the other properties will be ignored.
- "Up-to-Ignore" only applies for internal files of ROCKEY6 SMART. It represents an executable file that can only be executed on the condition that the system security level is lower than or equal to the executable file security level. Otherwise the executable file cannot be executed.
- "Internal" means that if it works for a data file, the data file can only be accessed by the
- ROCKEY6 SMART's internal executable files. It can be deleted by the external program only after successful super password verification. However in either case, it cannot be read or written by the external program. If it works for an executable file, the executable file contains hidden properties. If a user lists all directory files without supplying successful super password verification, all executable files with hidden properties cannot be shown. This is an effective way to protect executable files.
- Only after successful super password verification can executable files and internal files are created by using the API.

For implementation, use 1-bit high 4-length to represent the properties.

Table 4-2 Flags and Macros for File Properties

Property	Flag	Macro
Normal	0x00	FILEATTR_NORMAL
Execute	0x10	FILEATTR_EXEC
Directory	0x20	FILEATTR_DIR
Up-to-Ignore	0x40	FILEATTR_UPIGNORE
Internal	0x80	FILEATTR_INTERNAL

4.7 Security Mechanism

Core of the security mechanism is that based on the "system security level" which has 16 different levels. The combination of the "system security level" and the "file security level" control access permissions to all the files on the card. The COS will only perform operations on a particular file if the system security level is greater or equal than the security level set for the particular file. There are two different techniques to change the system

security level: verify the super password or perform an executable file on the virtual machine.

The COS security mechanism includes three sub-systems. They are:

4.7.1 Global Security

One 8-byte "super password" is adopted as the global security policy. Feitian, the ROCKEY6 SMART manufacturer, initially sets and provides the super password to the software developer. The super password can be modified or abandoned by the software developer. After successful verification of the super password, the "system security level" is set to its maximum value and all normal (excluding internal) data files in the card can be read or updated. Executable files can only be selected or deleted. The super password is always required for creating the executable files. *If the super password is set to all zeros, the developer can no longer modify or verify it* -this operation disables the super password and permanently eliminates access to super user privileges. Be careful before invoke this operation.

4.7.2 File System Security

There are two parts to the file system security: (1) file classification application, which allocates the files into different kinds of groups. The security levels of the groups are independent. Furthermore the file classification limits the program in the virtual machine by not allowing access to other files in different classifications. (2) On the other hand, the executable files must have a security level that is greater or equals to any data file that it attempts to run.

4.7.3 COS Security Limits

When a running executable file needs to change the "system security level", COS will limit the system security left to equal or less than the program security level.

Chapter 5. Remote Update Management

Dongle was used to protect software program by SW vendor with general acceptance; on the other side, owing to being a physical goods, the selling and updating of the hardware dongle can only be sent by post but through Internet, which is available for the pure software product. It is quite imperative, in such case, to achieve the function of remote update with the aid of hardware dongle.

Three types of remote update were integrated into the ROCKEY6 SMART, which are Update Tag (UT), Update File (UF), and Module Manager (MM).

Update Tag means to change the remote update tag of client-side dongle with secure method, intended to make internal program on client-side could manage to handle different actions based on the judgment of this tag; else to implement update combined with other two methods described below.

Update File implies to a full procedure of encryption/decryption across the server-side and client-side, particularly, encrypts the "plain-text file" first and sent it to user, and on the client-side, restore the "Cipher-text file" inside of the dongle back to the original "plain-text" one. Within such scheme, user may replace any file inside of the ROCKEY6 SMART in the direct way, which is quite powerful and popular for use, and was described later with stress.

Module Manager focus on executable files only, the SW vendor, recurring to some secure methods, may control the running status of one or a set of .exe file on client-side correspondingly; secure file transfer, in a similar way, can be used to implement this feature.

5.1 Remote Update Management

Update Tag (UT) is a basic feature that may be utilized by ROCKEY6 SMART developers to implement remote updates of their software protection and licensing systems. "UT" is based on "one time passwords", that is to say, key of each update operation is unique and can only be used once.

There are two main effects for UT: a) modify the flag at the client side to control the execution of program; and 2) increase a remote update password permission temporarily for use by 2 alternative solutions.

The "UT" operation involved several steps that are described as follows:

1. The software developer must first initialize the ROCKEY6 SMART dongle to set the Initial Remote Update Tag (IRUT) and Initial Remote Update Password (IRUP). Here, we suggest you to set initial tag and password for later use even if you do not plan to use this feature for the moment. The IRUP cannot be set as a single "F" in all size; similarly, a single "0" in all size is unacceptable for the IRUT value input. See Figure 5-1 for details:

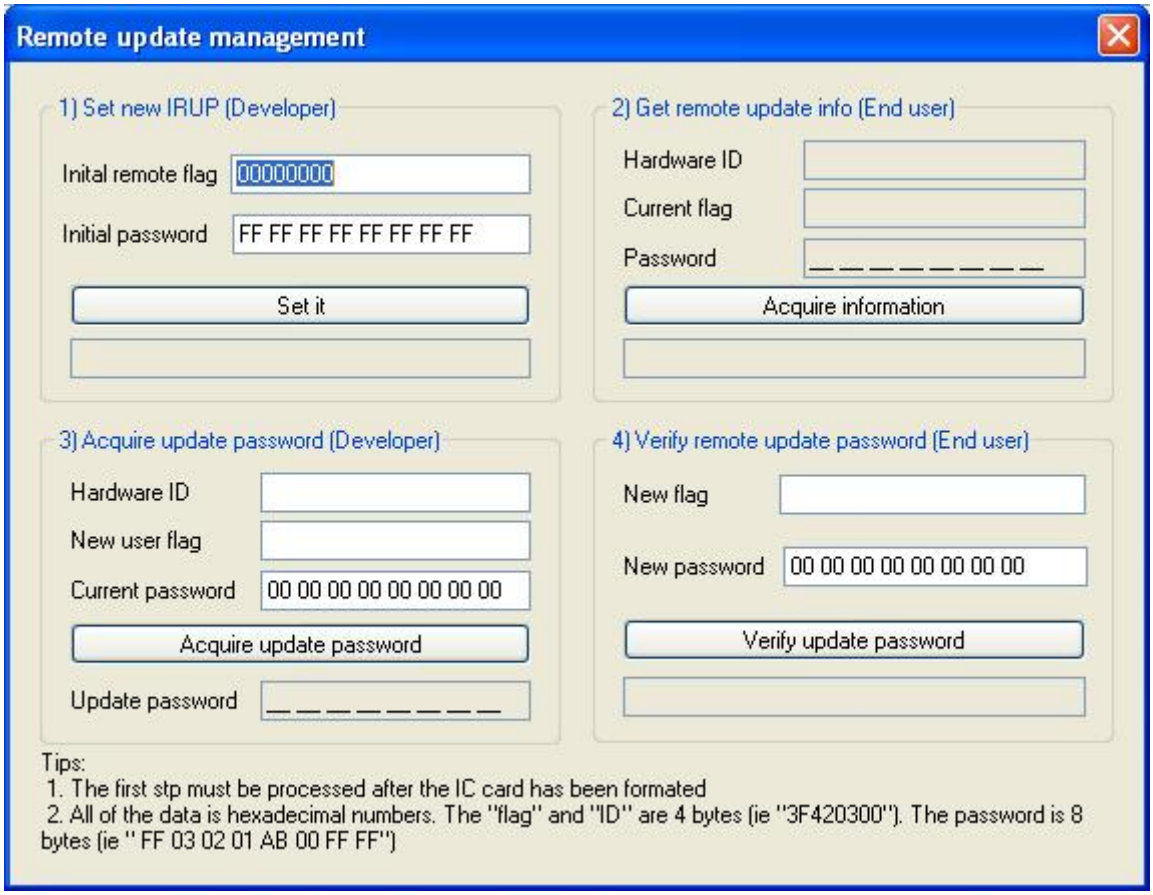


Figure 5-1

This step is the initial step of SW vendor before handing the ROCKEY6 SMART to the end user; in addition, since it's a one-time-only operation, the reset of IRUP is based on the reformat of ROCKEY6 SMART in a grain. For instance, we set IRUT as "0x00000001" and IRUP as "11 11 11 11 11 11 11 11" respectively, and it is recommended for you to use a more complex hexadecimal-string in the practical application.

2. When user needs to update software in a remote way, it is necessary for them to provide hardware-related information to the SW vendor, which including the hardware serial number, current tag, and current password, except the condition that all of these data had been recorded by SW vendor already.

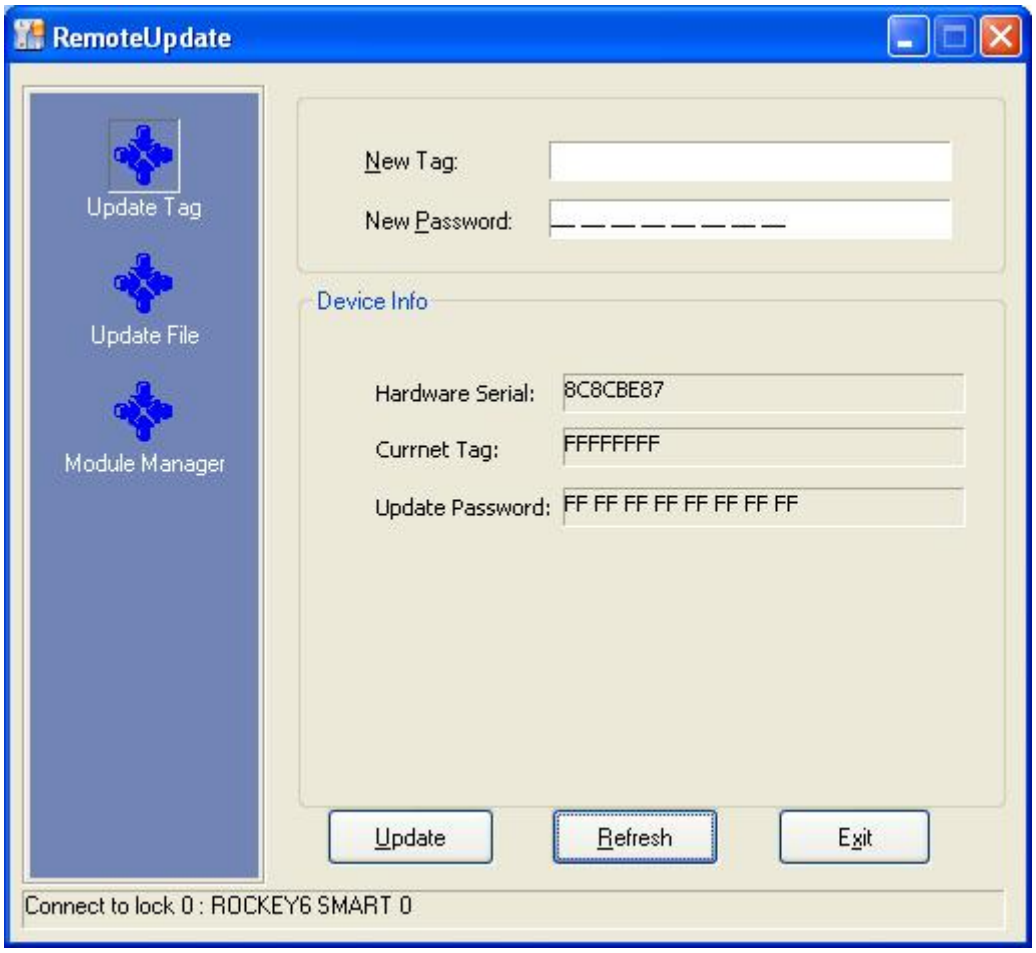


Figure 5-2

From the second step of this example, user may obtain all relevant information of his dongle by using "Remote Update" Tool, and send these back to the SW vendor by all means; otherwise, SW vendor may obtain this information relying on the code segment below if it is required to integrate the remote update feature into those applications of their own.

```
unsigned char RemotePass[8];
//Open the dongle
errcode = DIC_Command(hic, GET_REMOTE_INFO, cmddata);
RemoteTag = DIC_Get(cmddata, REMOTE_TAG, BY_VALUE, NULL);
DIC_Get(cmddata, REMOTE_PASS, BY_ARRAY, remotePass);
//Get the hardware ID of the current dongle
errcode = DIC_Command(hic, GET_HARDWARE_INFO, cmddata);
HardSerial = DIC_Get(cmddata, HARD_SERIAL, BY_VALUE, NULL);

m_HardSerial.Format("%08X", HardSerial);//Format the string of hardware serial number
m_RemoteTag.Format("%08X", RemoteTag);//Format the string of remote update flag
//Format the string of remote update password
```

```
m_Password.Format("%02X %02X %02X %02X %02X %02X %02X %02X", RemotePass [0], RemotePass [1] , RemotePass [2], RemotePass [3],
RemotePass [4], RemotePass [5], RemotePass [6], RemotePass [7]);
//Close the dongle
```

3. The software developer generates a remote update password. One thing should be noted that, if the top digit of the remote update tag is "1" (the first character greater than "8"), represents the "user update password" is correlated with the "hardware serial number"; therefore, all passwords generated will be unique to each other even with the same user tag altogether. On the other side, the "user update password" is independent with the "hardware serial number" that user input if the top digit is "non-1" value of the remote update tag.

4. Received "new tag" and "new password", user may start to process remote update in this way, and the "update tag" inside of the ROCKEY6 SMART will be replaced with the new one after update successfully. This step can be implemented by using the "Remote Update" tool provided on SDK, or you can achieve it in your application with following code segment embedded.

```
// Set the update flag to be the new remote update flage generatd by the developer（0x02）
DIC_Set(cmddata, REMOTE_TAG, BY_VALUE, 0x02, NULL);
//For simplified description, all new passwords are set to 0x22
memset(RemotePass,0x22,8);
DIC_Set(cmddata, REMOTE_PASS, BY_ARRAY, 0, RemotePass);
//Verify the remote update flag and password
iRet=DIC_Command(Hic,CHECK_REMOTE_INFO,&rInfo);
```

By above steps, the flag in the dongle is replaced with the specified flag. If you use only this solution to update, you can pre-set an algorithm into the dongle before it is delivered to end users, and make determination with an internal program as follows:

```
dword dwTag;
//Get the remote update flag
get_remote_tag(&dwTag);
if(dwTag & 1)
    ;//Do something
esle if(…)
    ;//Do something else
else
…
```

Now, you may have known how to place an algorithm into the dongle.

5.2 Secure File Transfer

The ROCKEY6 SMART dongle supports Remote Update Management (RUM). The Remote Module Management (RMM) function will verify the user's identity, but it cannot update the external program because the end user does not have access to the super password, which is required to write to the dongle.

The SFT process allows the software manufacturer to securely update external programs or data files without compromising the super password.

The update procedure is as follows:

1. The SW vendor creates a "Cipher text file" with their ROCKEY6 SMART dongle
2. The manufacturer sends this "Cipher text file" to their customers
3. The customer transfers the "Cipher text file" to the dongle with the program provided by the SW vendor. The dongle will decrypt the "Cipher text file" and create a "Plain text file".

The full process of implementing "Update File" can be referred to the Figure 5-8.

Supposed that the name of your executable file is "0x1000" and ID is "0x0001", there are three solutions for you to update file with "update file" function, in combination with "update tag" information described above.

- ✓ Solution 1: All users are same

Introducing by the feature of "Remote update management", if no new password for remote update was generated after ROCKEY6 SMART plug-in, the "Cipher text file" created by SW vendor can be looked as one applied-to-all; in that way, any user belonging to the same SW vendor has no need to verify the remote update password to finish update operation overall (cannot be verified actually, it is required to re-plug in the dongle if the password was verified already).

This solution is quite simple for SW vendor to implement without considering specific process of remote update, just following such steps as: use IDE to encrypt the executable file with "Generate cipher-text file" function, and send this "cipher-text file" (file output from the function of "Generate cipher-text file") to end-user.

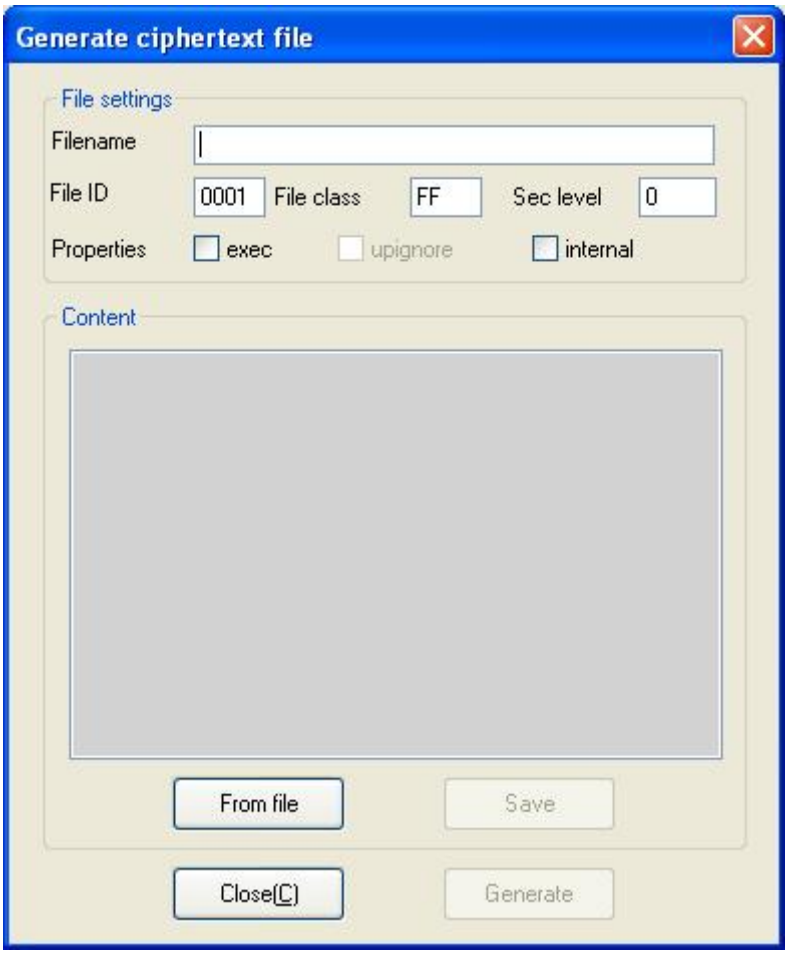


Figure 5-3 Generating Encrypted File

While receiving this "cipher-text file", the user needs to decrypt it with "Generate plain-text file" function, and keep this "plain-text file" storing inside of the Rocky6 Smart. After that, updated software can be utilized by user in the normal way; moreover, user can complete the full process of remote update with the help of "Remote Update" tool.

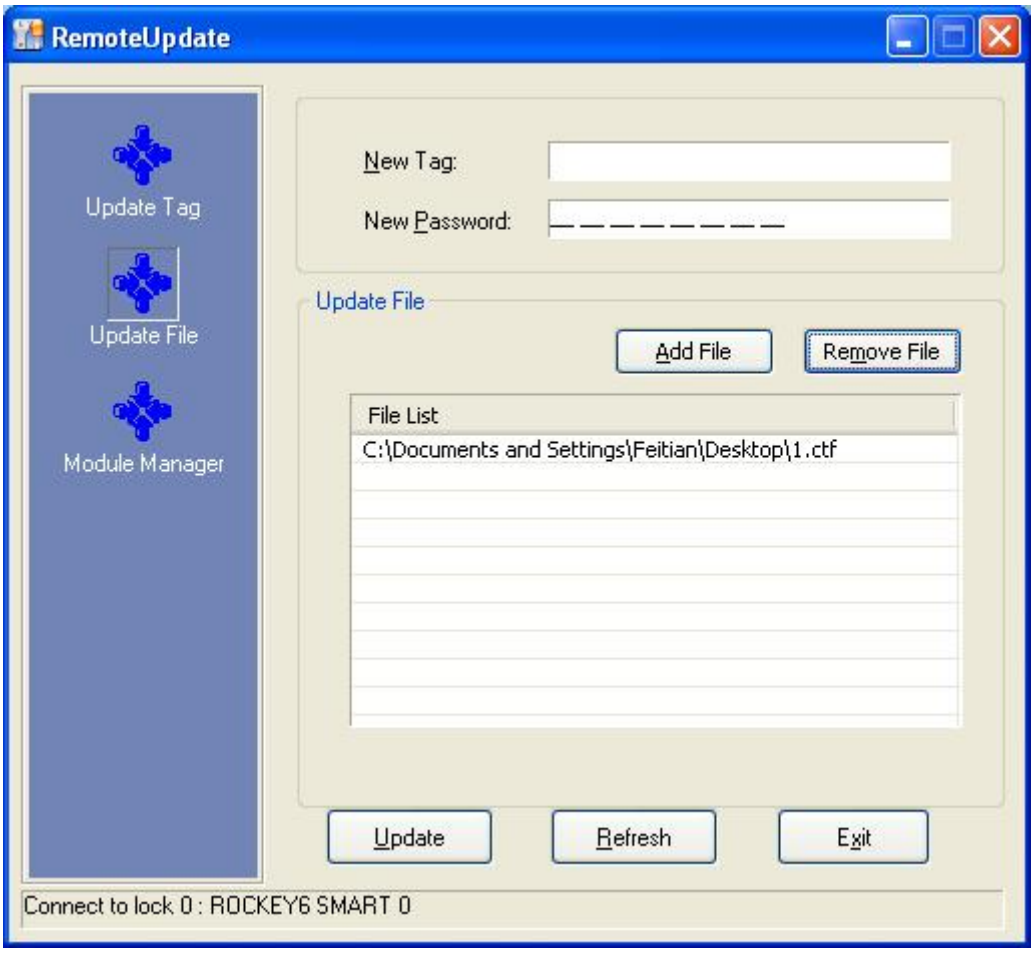


Figure 5-4 Using Tool For Update

If SW vendor plans to integrate this remote update feature with applications of their own, understanding the "PLAINTEXT_FILE" macro is a prerequisite, which was used to generate plain-text file inside of the ROCKEY6 SMART. Specifically, pass this macro to "DIC_Command" at first, and make the memory of plain-text file pointing to the "cmddata" variable, a plain-text file will be generated inside of the dongle in this way, see code segment below:

```
memcpy(cmddata, c_exefile, exesize);
errcode = DIC_Command(hic, PLAINTEXT_FILE, cmddata);
```

Note: "c_exefile" is the buffer stored cipher-text file, "exesize" is the size of cipher-text, "hic" is the handle of ROCKEY6 SMART which was opened without super password verification.

✓ Solution 2: Part of User is same

During the initialization process of remote update tag and password, set the "Initial remote flag" and "Initial password" as the same value for part of user and make sure the top digit of "Initial remote flag" as "0", thus SW vendor may process update respecting a specific set of user, making sure the update file are same for all of them.

Using this solution, it is necessary to take consistent manufacture-settings respecting "Initial remote flag" and "Initial password" to the same type of Rocky6 Smart in a batch mode; here, we supposed to set both of two settings as "1" for the simple explanation.

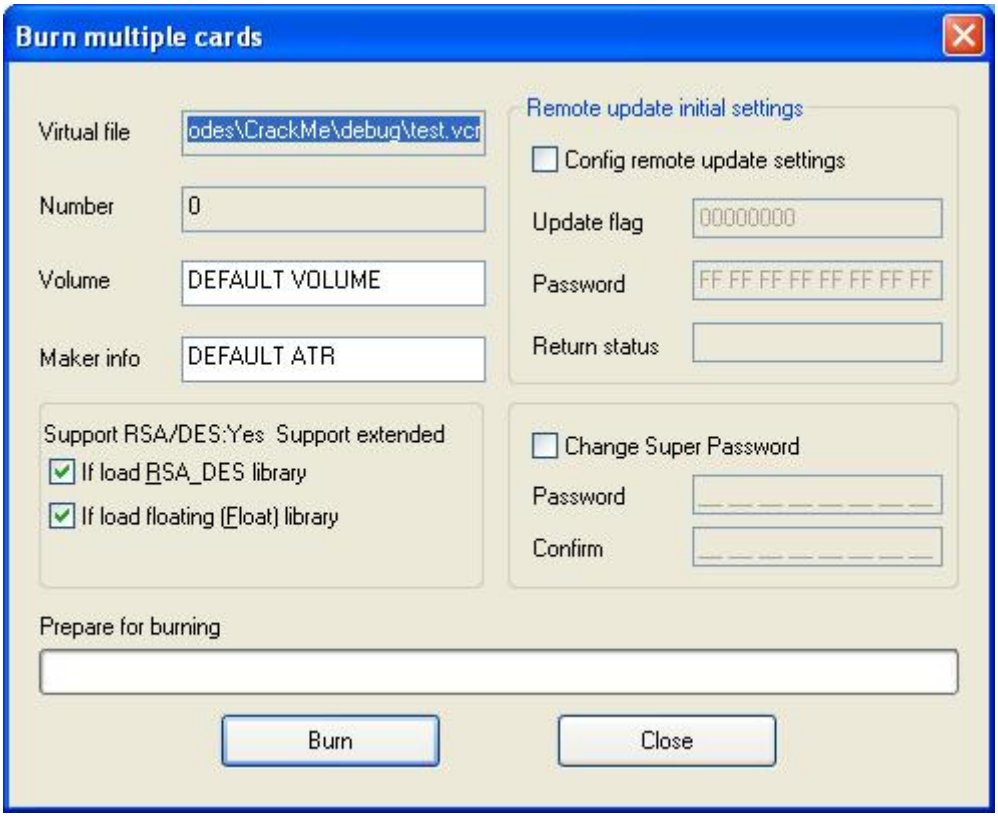


Figure 5-5 Setting "Initial remote flag" and "Initial password" when burning in batch mode

After successfully setting "Initial remote flag" and "Initial password", SW vendor can send these ROCKEY6 SMART dongles to their end user. Following steps are necessary to implement remote update:

1. Enter a "New user flag" and "Current password" against the third step ("Acquire update password") on the main screen of "Remote update management", here we take "22222222" as the "New user flag". Clicking the button "Acquire update password" will generate the "Updated password" and appear in the field below, keep information of "New user flag" and "Updated password" in safe place for later use.

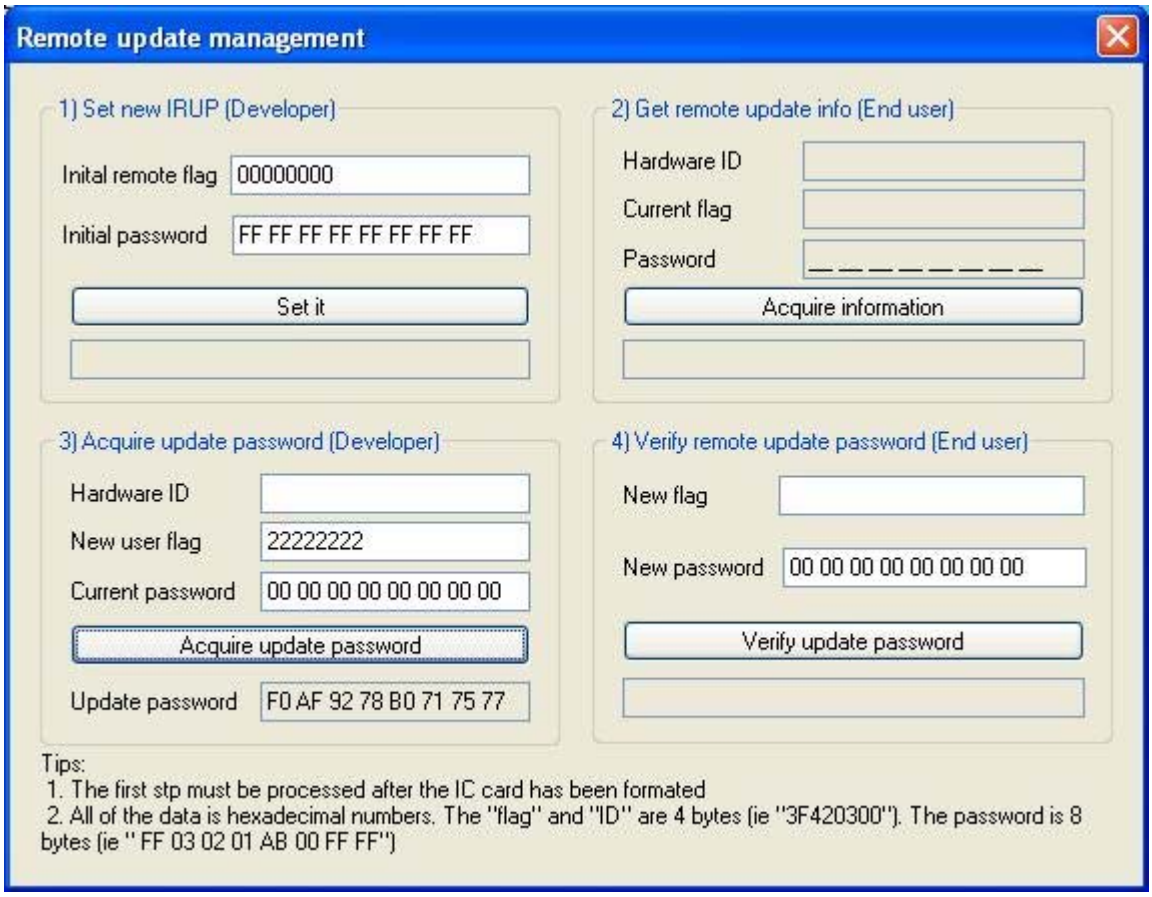


Figure 5-6

2. Return back to the main screen of "Real device" of IDE, click option on menu to generate Cipher-text file, which involves relevant information about remote update. At such case, SW vendor can send their user all of this information, including "New remote flag", "Updated password", and "Cipher-text file" just generated.

3. Received these information, user may update at client-side based on the tool of "Remote Update". Enter the "New Tag" and "New Password" on the "Update Tag" tab screen in first step, and click "Add File" button to select specific files to add into the "File List" show below.

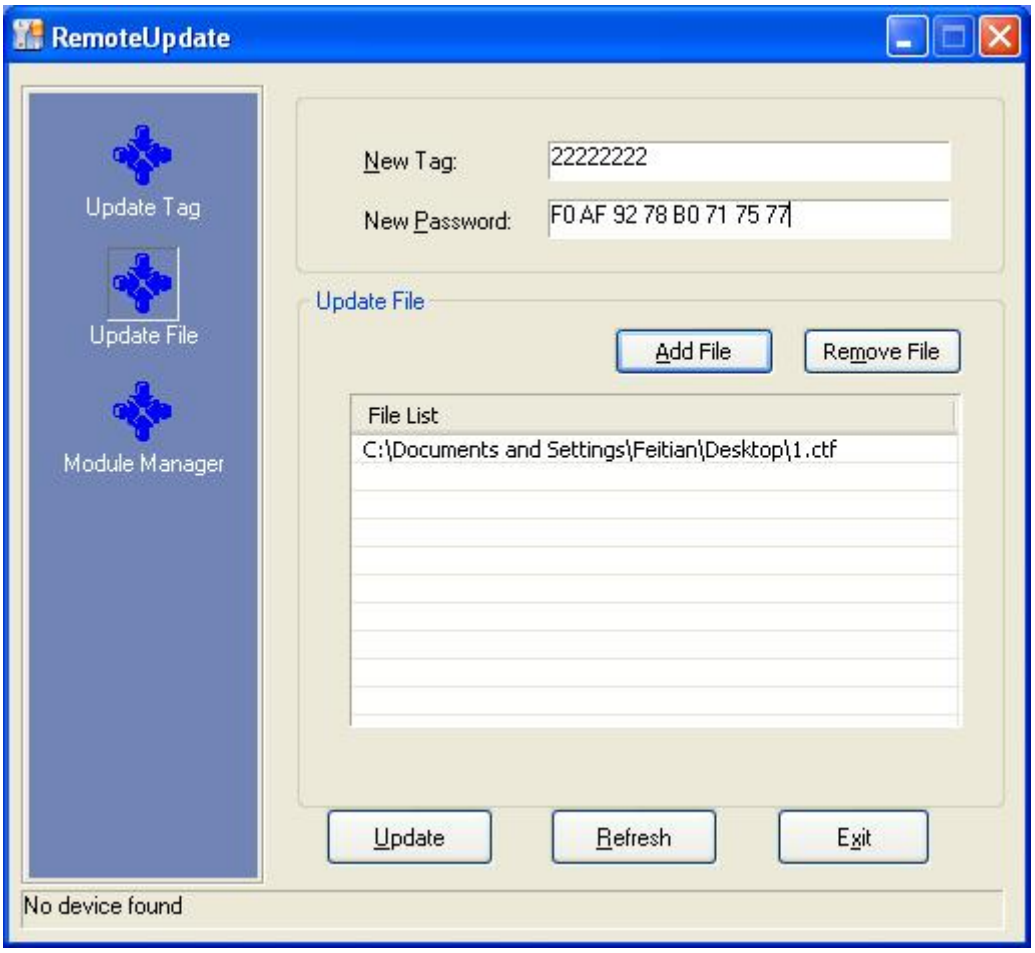


Figure 5-7

The remote update process, at the end, is completed in full-range.

✓ Solution 3: User unique to each other

If setting the top digit of "Initial remote flag" as "1" during the period of 'set New IRUP", that is to say, the value of "Initial remote flag" greater than "0x8000000000", the "Updated password" generated by remote update is relevant to the hardware serial number of Rocky6 Smart, thus making sure "update files" of all user are unique to each other. Comparatively speaking, this solution and "Part of User is same" solution are about equal except that the "Initial remote flat" is set to "1", and more important, SW vendor have got to provide each user the only "update files". Due to the large number of user, SW vendor could not update user by manually operating tools, but with the help of program running. During manufacture-initialization, it is approved to apply the "Burn real card" function of the first solution that all "Initial remote flag" and "Initial password" of ROCKEY6 SMART will be set to the same value, with the condition that the "Initial remote flag" had not been used elsewhere.

While updating, SW vendor generate cipher-text file with some basic information obtained by user.

Before writing code, it is a must for SW vendor to understand the "CRYPTOTEXT_FILE" macro, which is used by ROCKEY6 SMART to complete the operation of "Generate Cipher-text file". The command buffer required by this macro includes a "DISCT_File" in the forefront and data of the file next to it. The macro is used in the same way with the process of generating a file and writing data inside; whereas, it is a one-pass operation by "CRYPTOTEXT_FILE", and store the generated cipher-text file into the buffer pointed by the variable of "cmddata". All important data can be copied from the buffer to a file and kept in safe mode for later use.

Operation to obtain "Initial remote flag":

```
// Set new update flag to 0x02
DIC_Set(cmddata, UPGRADE_REMOTE_TAG, BY_VALUE, 0x02, NULL);
//Provide the hardware serial number of the dongle (0x7A85728C used as example here)
DIC_Set(cmddata, UPGRADE_HARD_SERIAL, BY_VALUE, 0x7A85728C , NULL);
memset(buffer, 0x02, 8);//Set all new passwords to 0x22
DIC_Set(cmddata, UPGRADE_REMOTE_PASS, BY_ARRAY, 0, buffer);
errcode = DIC_Command(hic, GET_UPGRADE_REMOTE_PASS, cmddata);
//Put new remote update password into RemotePass
memcpy(RemotePass, cmddata, 8);

DIC_Set(cmddata, FILL, 512, 0, NULL);   // Clear
            DIC_Set(cmddata, FILE_ID, BY_VALUE, 0x0001, NULL);// File ID
            DIC_Set(cmddata, FILE_CLASS, BY_VALUE, 0xff, NULL);// File class
      // File properties
DIC_Set(cmddata, FILE_ATTRIBUTE, BY_VALUE, FILEATTR_EXEC, NULL);
// File size
DIC_Set(cmddata, FILE_SIZE, BY_VALUE, READDATA_SIZE, NULL);
            // Name of executable file is 1000
DIC_Set(cmddata, FILE_NAME, BY_ARRAY, 0, "1000");
DIC_Set(cmddata, FILE_DATA, BY_ARRAY | READDATA_SIZE, 0, (char*)g_progReadData);// File content
//Generate a plain file
            errcode = DIC_Command(hic, CRYPTOTEXT_FILE, cmddata);
            exesize = DIC_Get(cmddata, FILE_DATA, BY_ARRAY, c_exefile);
```

Note: Bothe "cmddata" and buffer are buffer with enough space, "RemotePass" is an 8-byte array, "c_exefile" is used to store the returned cipher-text file, hic is a handle opened with super password verification, and "READDATA_SIZE" is the size of plain-text file.

Following code segment can be used by SW vendor, in programs running on the client-side:

```
// Set the update flag to be the new remote update flag generated by the developer（0x02）
DIC_Set(cmddata, REMOTE_TAG, BY_VALUE, 0x02, NULL);
```

```
//For simplified description, all new passwords are set to 0x02
memset(RemotePass,0x22,8);
DIC_Set(cmddata, REMOTE_PASS, BY_ARRAY, 0, RemotePass);
//Verify remote update flag and password
iRet=DIC_Command(Hic,CHECK_REMOTE_INFO,&rInfo);
        memcpy(cmddata, c_exefile, exesize);
        errcode = DIC_Command(hic, PLAINTEXT_FILE, cmddata);
```

Note: cmddata is a buffer with enough space, RemotePass is an 8-byte arrary, c_exefile is the buffer with cipher-text file storing inside, exesize is the size of cipher-text file, hic is the handle of opened ROCKEY6 SMART.

A point should be taken in consideration respecting above three solutions, the operation of 'secure file transfer" and "remote update tag" is operating concurrently if solution 2 and solution 3 were applied. Further speaking, SW vendor set the "New tag" and "New password" to their ROCKEY6 SMART based on the dongle-related information from the client-side; with this "New password" at hand and "Updated password" saved already, process "Generate Cipher-text file" and send user above three files altogether; after that, user may generate plain-text file on the condition that the "New tag" and "New password" had been verified successfully. Mind out the SW vendor's operations of obtaining new password and generating cipher-text file are in the same time, dongle plugging-out or re-opening is forbidden in the midcourse; in the same manner, the verification of new password and new tag on client-side are one-step operation, or all correlations between generated cipher-text file and remote update information are of no effect, like the solution 1 described before.

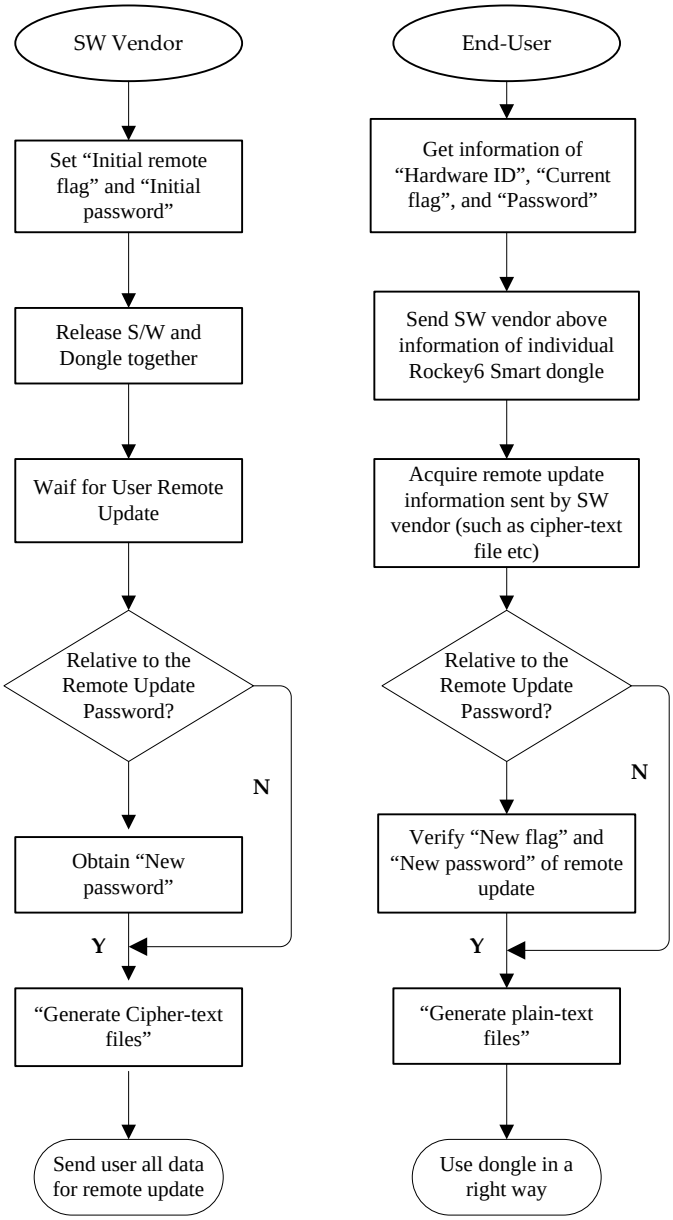


Figure 5-8

5.3 Remote Module Manager

Remote module manager includes three parts: setting module definition file, creating module authority file and setting dongle information module in bulk. For the ROCKEY6 SMART SW vendor, to manage the remote module requires two steps: firstly, define a module definition file using the module definition function; then sent it to the user. The users convert the module definition file to the module request file according to the actual module they purchased. Secondly, the SW vendor generates the module authorization file from the users module request file, and sends it back to the user again. As soon as it was received, user my start to update the ROCKEY6 SMART without delay based on the module authorization file. If SW vendor make clear of the required module files for each user, the first step can be completed all by SW vendor alone.

5.3.1 Module Definition

Choose and fill "Module definition" from operation types, and add it to the "file path" via a browser. Then click the "Add" button. A module window will pop up. Input the module name for "Module", and choose the dongle file. Click "OK" to add the module. Once back to the "Remote module management" window, click the "Add" button to add more modules. If more files need to be added into a module, all then that is required is to input the same file names into the "Add Module" message box. Once the module is added, click the "Open/Close" button to modify the state of the module. Finally click "Generate file" to create the module definition file.

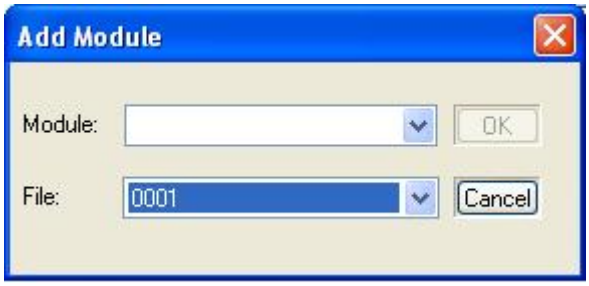


Figure 5-9 Adding Module

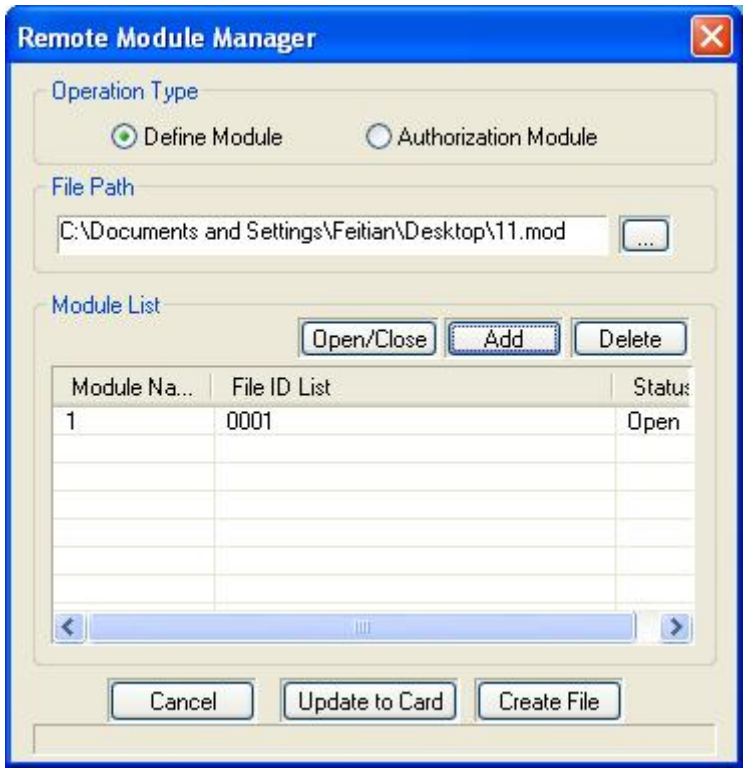


Figure 5-10 Remote Module Management

5.3.2 Module License

To generate a license file, the developer can convert it simply according to the received user request file. That is,

choose "Module authorization" from "Operation Class", then browse the files and add the user request file. After that, all user requests are listed. The developer can open and/or close any modules as required. Click "Generate file" to finish the module license procedure. The last thing is to send the generated file to the user for updating.

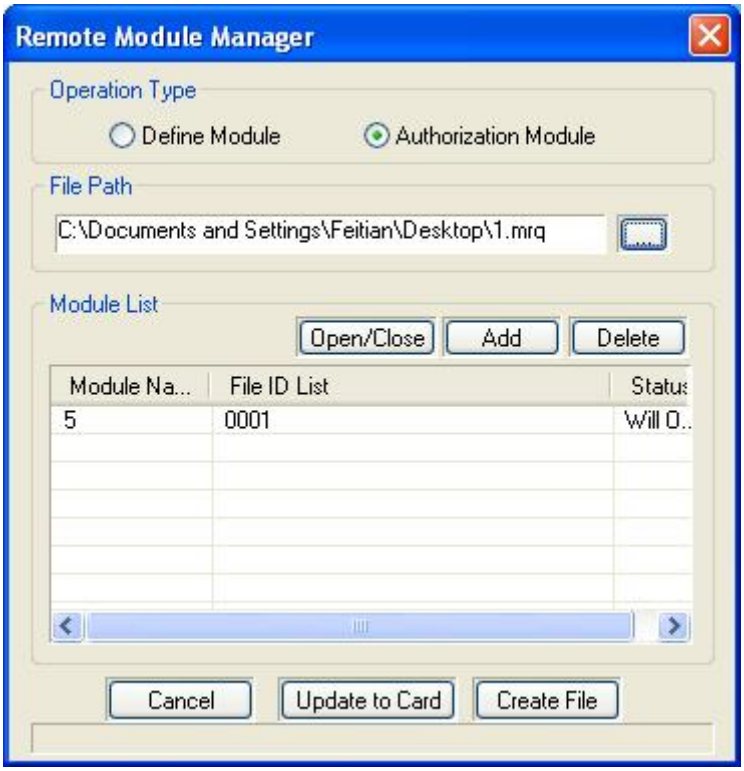


Figure 5-11 Module Licensing

Chapter 6. Production Management

Once you set up the C51 program, you can burn dongles in bulk. The steps are shown below:

Open ROCKEY6 SMART IDE, as shown in Figure 6-1.

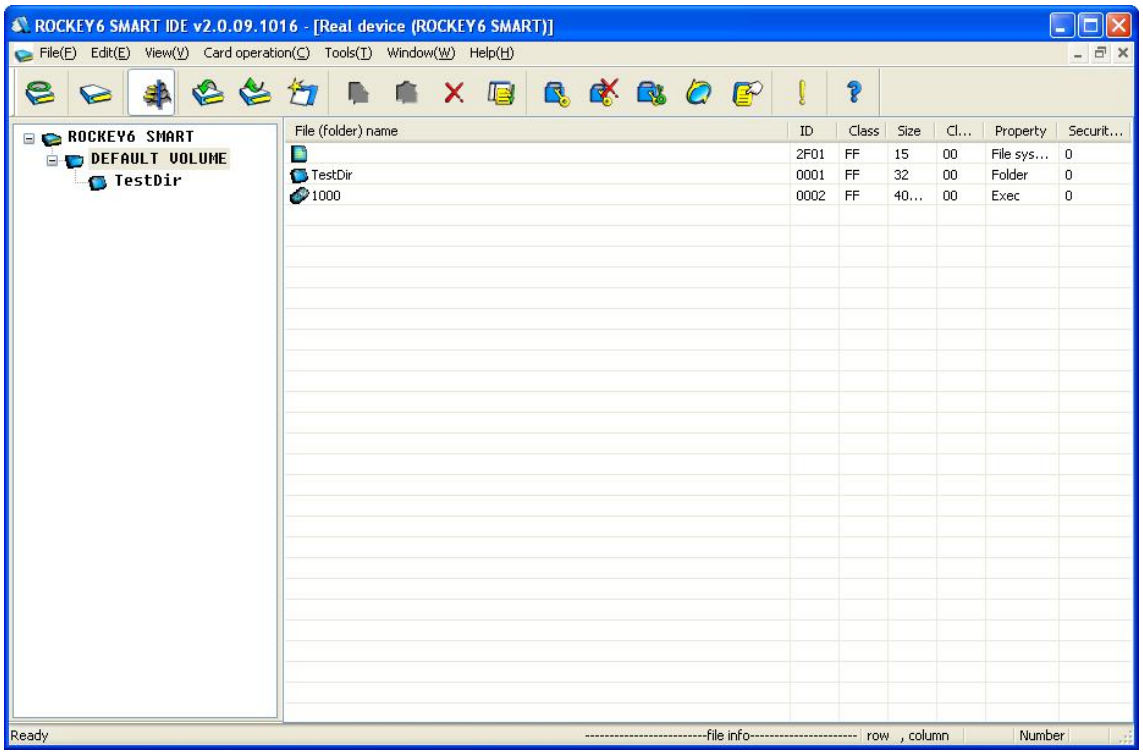


Figure 6-1 ROCKEY6 SMART IDE

Click the "File | Open Virtual Device" menu; then select the intended virtual card file (*.vcr) from the pop-up window and open it, as shown in Figure 6-2:

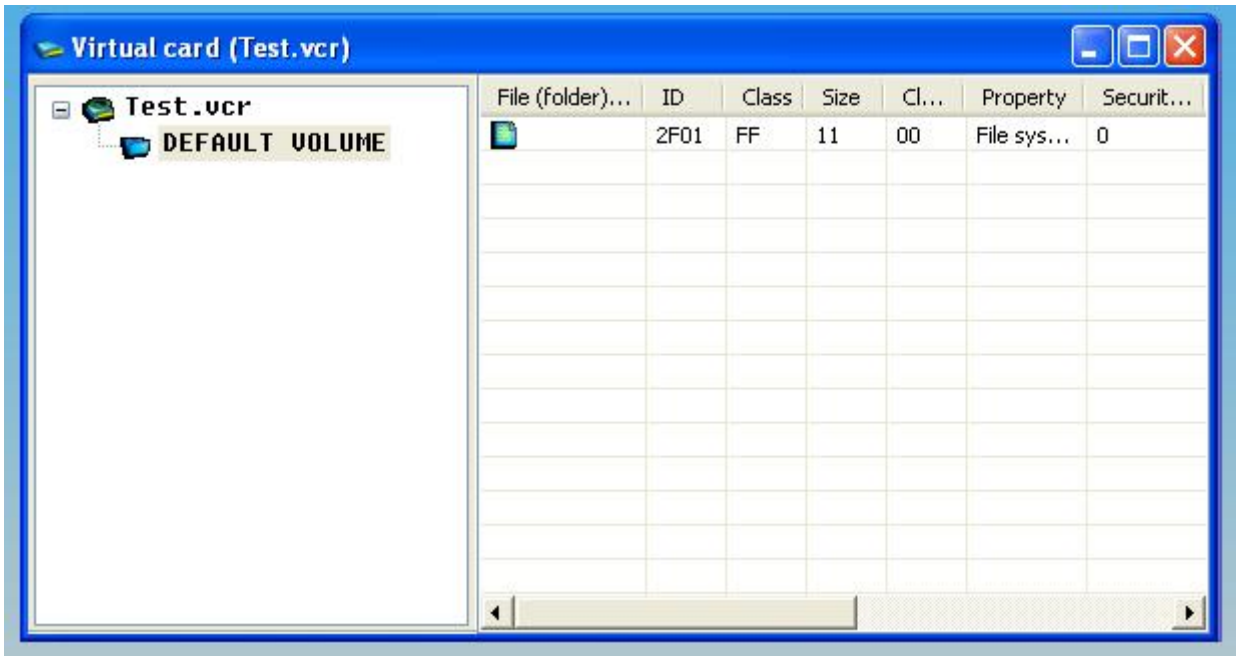


Figure 6-2 Virtual Device View

After choosing the executable file, right click your mouse, a menu will pop up as shown in Figure 6-3. Click "Burn", the executable file is selected as in Figure 6-4, and its icon is changed. When an executable file is selected to "Burn", it can be downloaded to the real card via the "Burn real card" function from the IDE. Otherwise, the executable file never downloads to the real card without selecting "Burn".

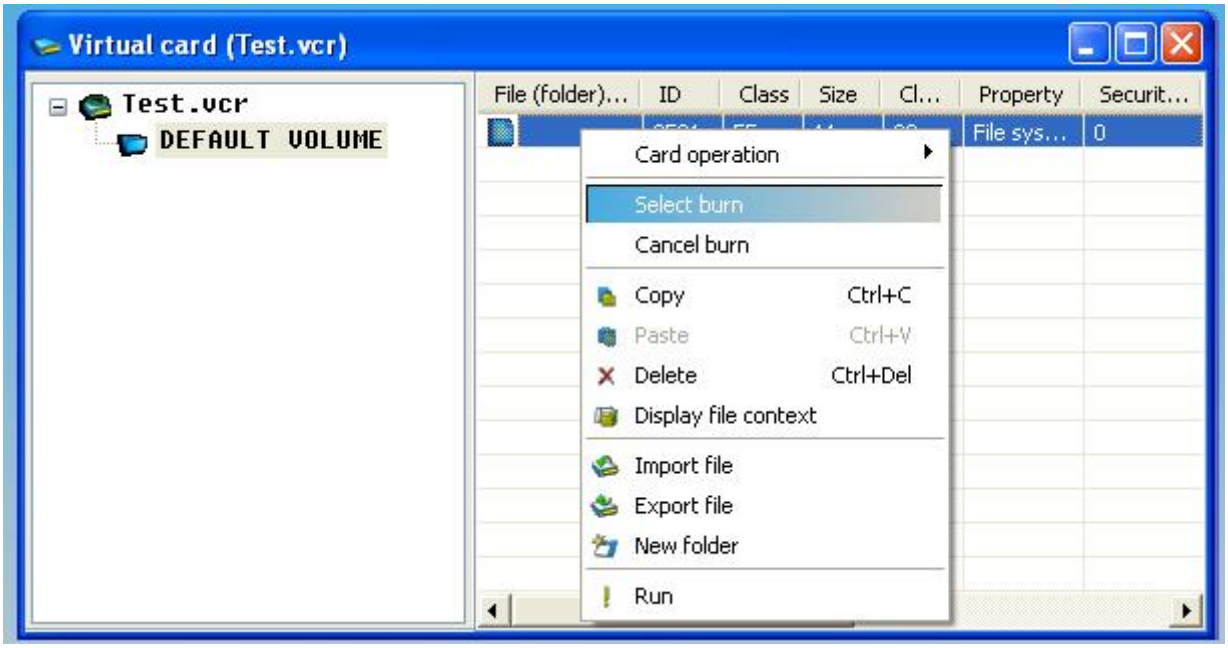


Figure 6-3 Choosing the program to burn

The last step is to burn the selected program to the real card. Choose "card.vcr", and right click your mouse. A menu will pop up as in Figure 6-4. Click "Burn to real card", then a "Burn in bulk" window will be shown.

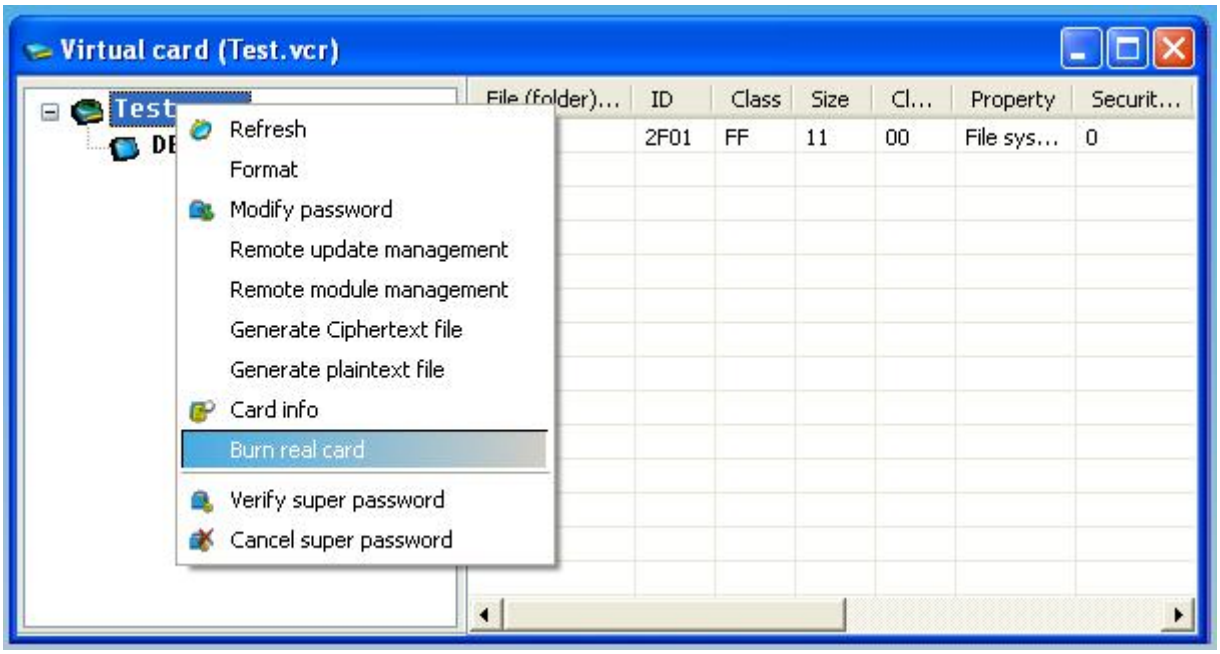


Figure 6-4 Burning to the real card

6.1 Burning in Bulk

The "Burn in bulk" window contains "Volume", "Manufacturer information" and "Remote update information". "Volume" is the name of the file system root directory in the dongle. "Manufacturer information" holds all relevant information about the manufacturer. "Remote update information" can be left without setting until the user really needs it. For details of remote update, please go to section "5.1 - Remote Update Management".

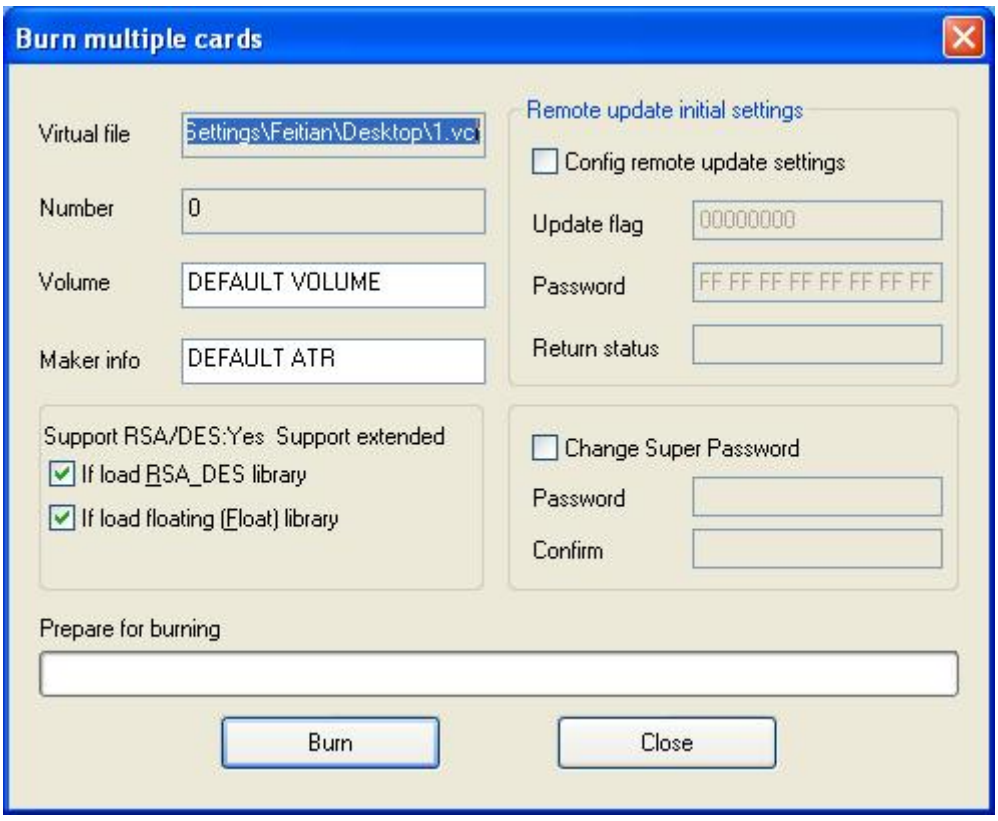


Figure 6.1-1 "Burn in bulk" window

When the real card is burnt, ROCKEY6 SMART can choose to download some library files as in Figure 6-5.

Once the burning process is finished, all ROCKEY6 SMART project information will be recorded into the "card.bfl" file. The user does not need to repeat the whole setting in the next burning process.

The "card.vcr" and "card.bfl" can be transferred to anybody who is going to burn a real card. All that is required is to follow the steps as in "Figure 6-4".

If a program contains a numbers of projects, and every project is allocated with a file ID and file name, several virtual devices should first be opened as in project (Figure 6-6). After that, copy all the executable files from every virtual device to one virtual device (Figure 6-7). Finally, give the virtual device with all setup executable files and "card.bfl" to the person who will do the burning process.

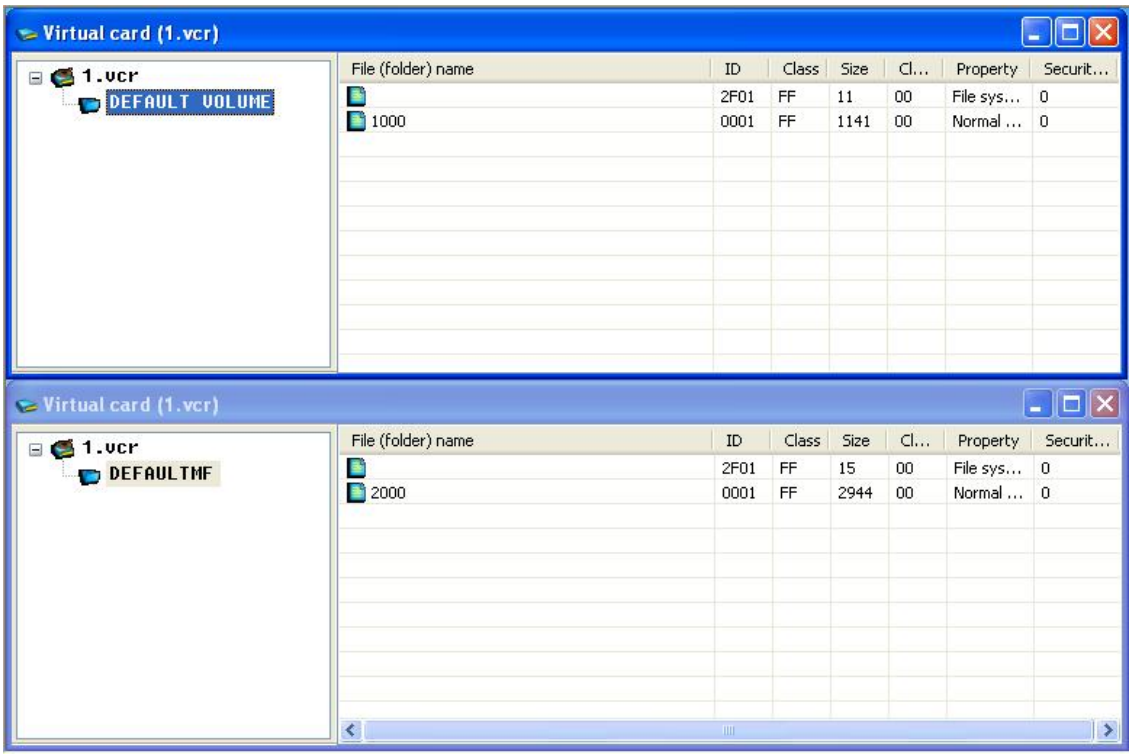


Figure 6.1-2 Openning multiple virtual devices in the same project

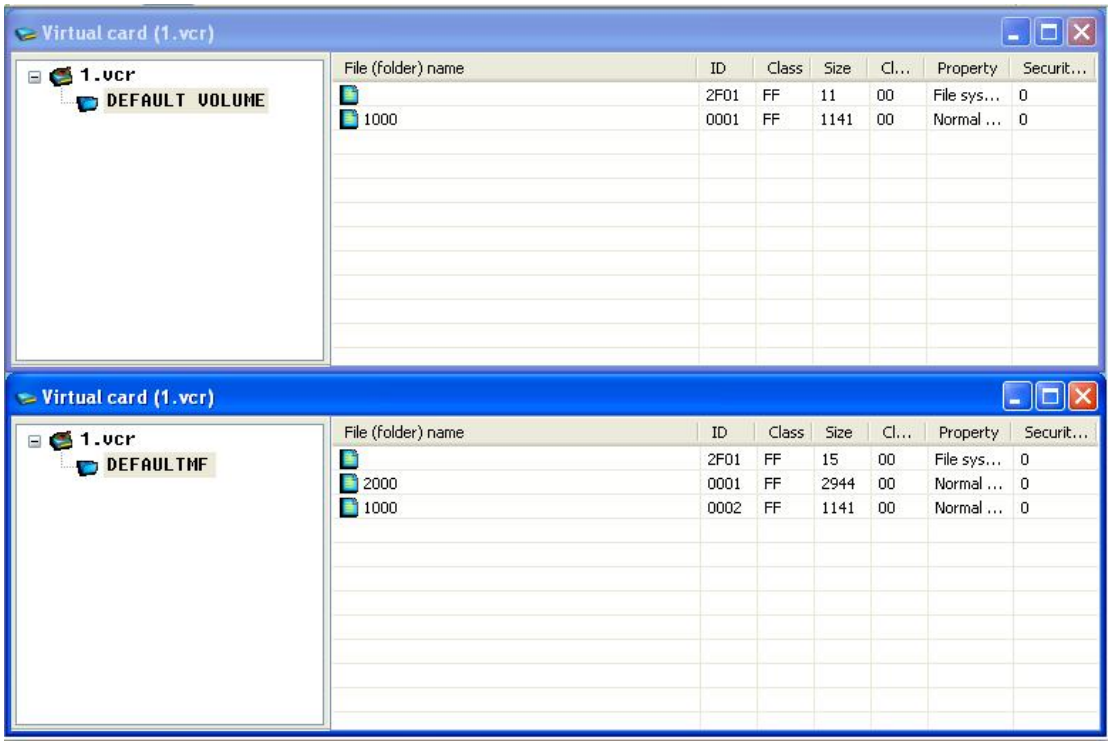


Figure 6.1-3 Copying the executable files to a virtual card

PART 2 Applications

After finishing this chapter you will know how to use Keil u Vision2 to compile the kernel of your encryption application and choose the corresponding API function, and you will also learn the development procedure of ROCKEY6 SMART.

Chapter7, Debug with ROCKEY6 SMART Simulator

This chapter will describe how to configure the development environment of Keil u Version2.After finishing this chapter the user will grasp some fundamental debugging procedures of Rocket6 SMART.

Chapter 8, Rocket6 SMART Essential

This chapter will show how to encrypt a program step by step from a simple sample. You will learn how ROCKEY6 SMART protects your applications.

Chapter 9, API Reference of Communication with ROCKEY6 SMART

The user will learn how to use the API from this chapter.

Chapter 10, API Reference

This chapter is the continuation of the last. The user will learn more details of the API from a sample.

Chapter 7. DEBUG WITH ROCKEY6 SMART SIMULATOR

ROCKEY6 SMART Simulator is used for simulating all hardware functions for ROCKEY6 SMART. The user does not actually need the hardware to perform the real card functions, as well as software debugging and execution.

Note: ROCKEY6 SMART Simulator is based on the Keil uVision2 debugger. Therefore, it can only work for code that is compliant with the Keil environment. If users do not need to debug the C51 program, it is not necessary to configure the simulator as well.

7.1 Configuring KEIL IDE

Before using Keil to develop ROCKEY6 SMART, some projects need to be set. Firstly, copy "\TOOLS\Debugger\RySSimulator.dll" from the ROCKEY6 SMART installation folder to "Keil\C51\BIN" and also copy "\API32\Dynamic\dic32u.dll" to "keil\UV2", or copy "DIC32U.dll" to the system folder. After that modify"TOOLS.INI" and append "TDRV4=BIN\RySSimulator.dll ("FEITIAN RockeySmart Simulator") "at the end of the "C51" block. "TDRV4" is the serial number and it can use the next number to replace the number when it is occupied.

7.2 Creating Project

To create a new project, open "Project" of KEIL UV2 and click "New Project". Input the project name in the pop-up dialog box and save it.

Choose 51 serial CPU when "Options for Target 'Target1'" appears. Users need to re-choose CPU for the existing project without 51 serial CPU. See Figure 7-1.

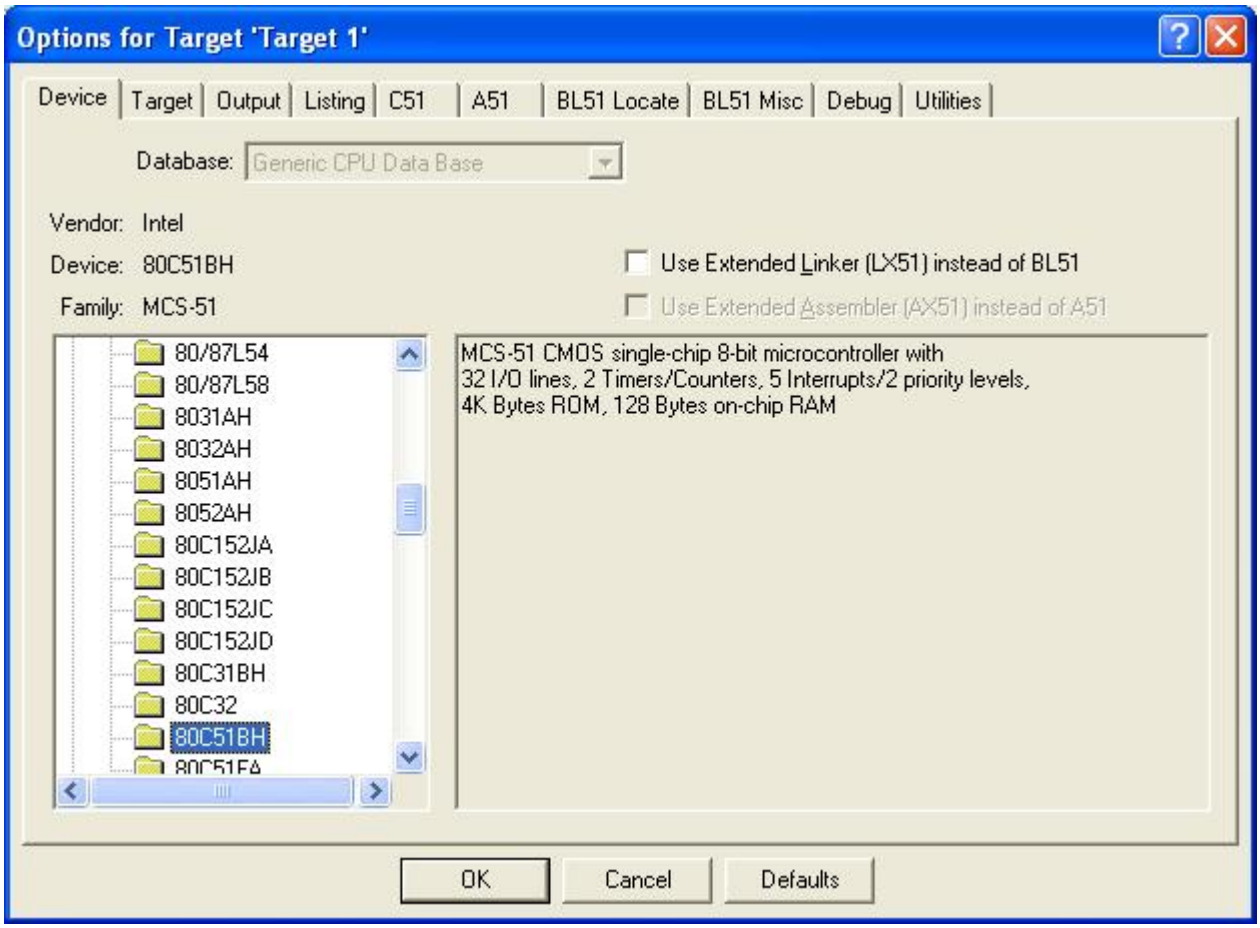


Figure 7-1

7.3 Setting Project Options

Click "Project" -> "Options for Target 'Target1'" to get high efficiency, use default setting (small model) in Target menu as shown in Figure 7-2.

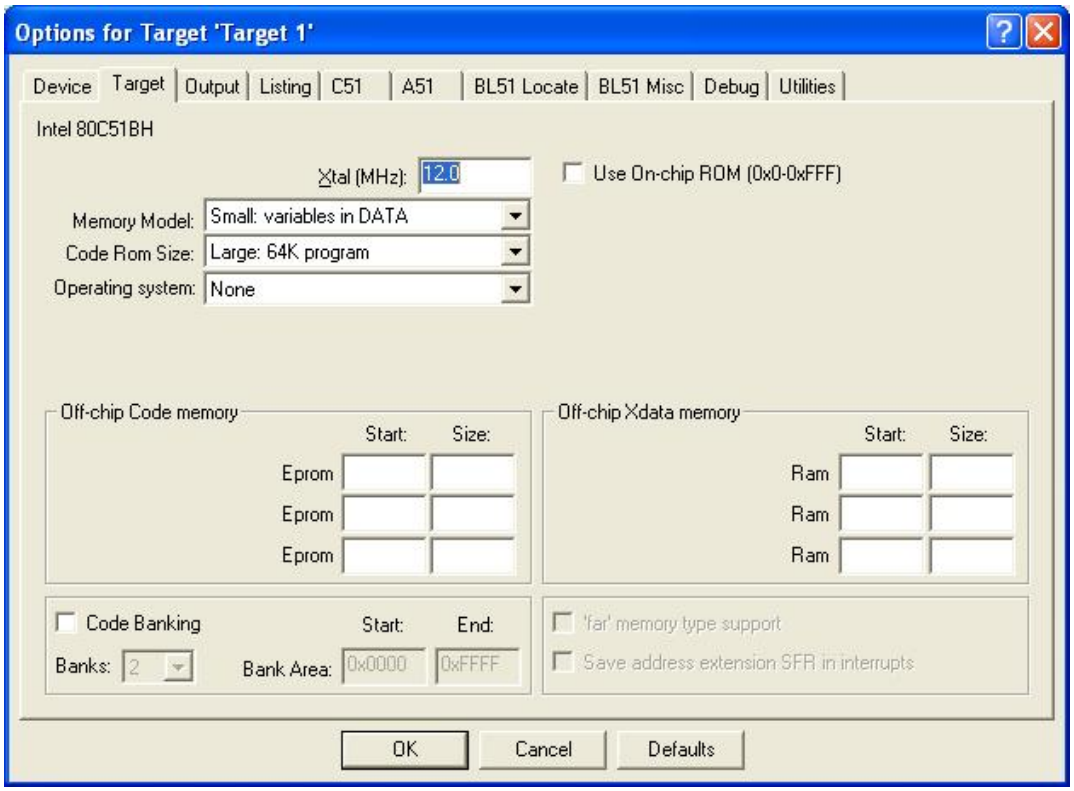


Figure 7-2

In Figure 7-2 "Project---Options for Target 'Target1' " page, tick "Create HEX File" and "Run User Program #1" and type "hexbin.exe test.hex test.bin" in the text box. Additionally the name of "test.hex" and "test.bin" will be changed according to the name of the project. Before using it, copy \TOOLS\Debugger\hexbin.exe to the project folder. Please note, it is not necessary that you use this option only for debugging your application; however it is necessary that your application is ready for bulk burning to ROCKEY6 SMART by writing extended code using third party tools, because the BIN file is an executable file in the ROCKEY6 SMART.

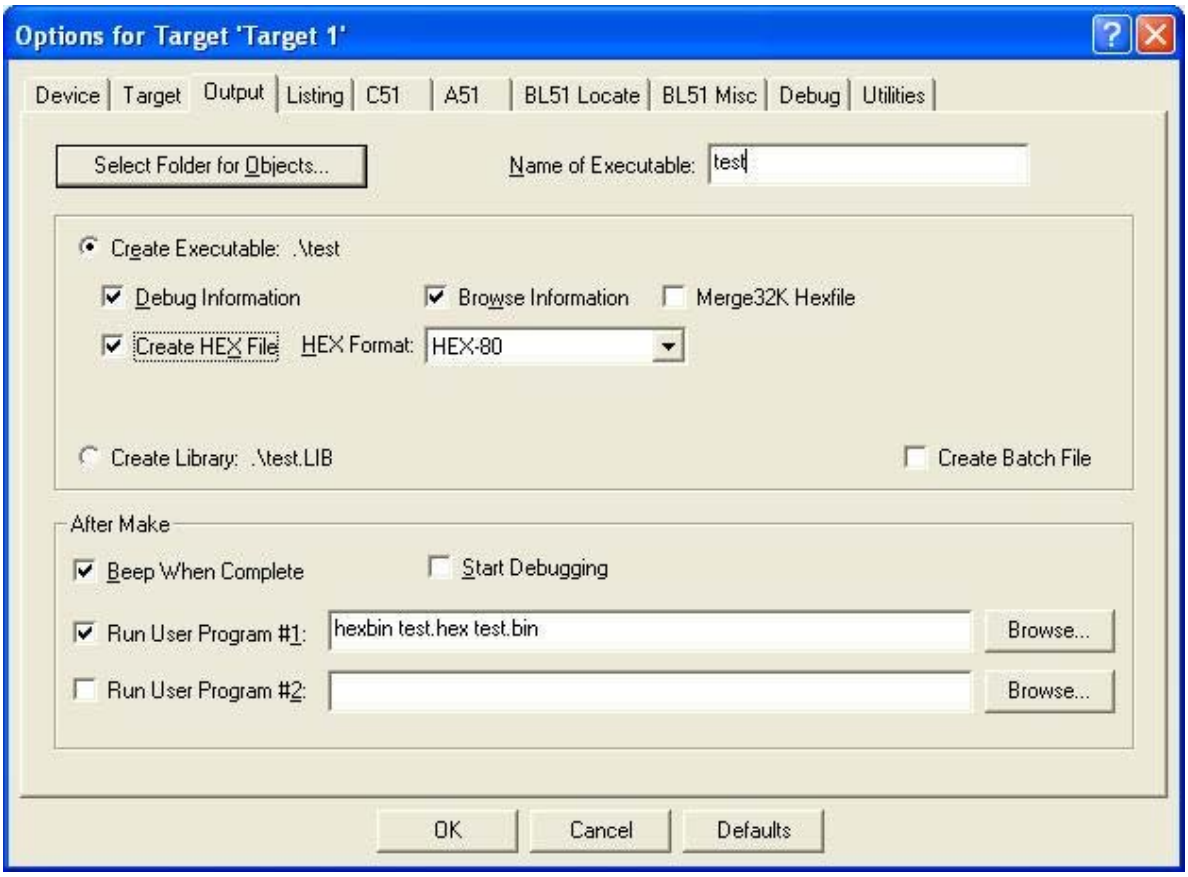


Figure 7-3

Select the simulator in the debug page and tick "Go till main". When programming the ROCKEY6 SMART and debugging the program is required, the software developer has to tick "Go till main". That is for jumping over the initial code and jumping into the main function. See Figure 7-4.

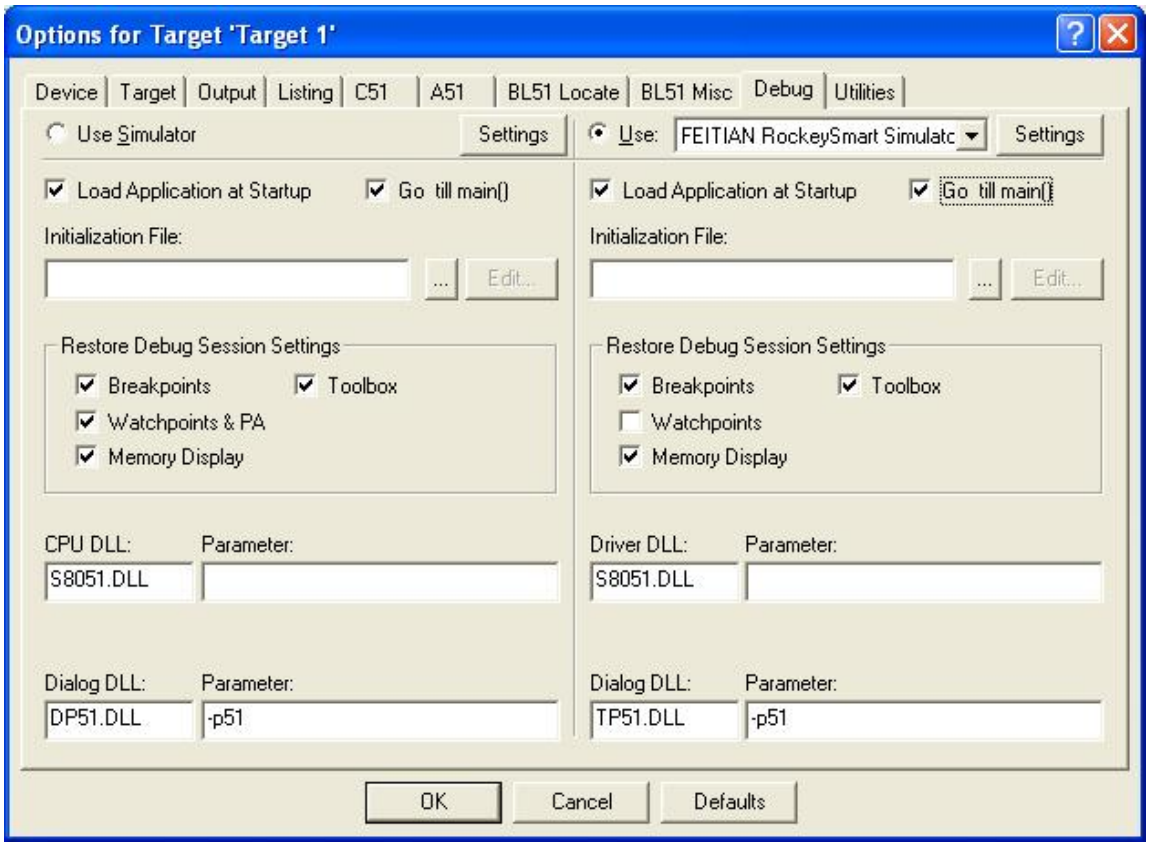


Figure 7-4

After this, click "setting" to configure the FEITIAN ROCKEY6 SMART simulator. If you connect the real card then select the "Using real card" option. Whether choosing this option or not is dependent on whether the user downloads the program into the real card to debug the program. Otherwise, the user does not need configure this option and also can even debug the program without a real card as shown in Figure 7-5:

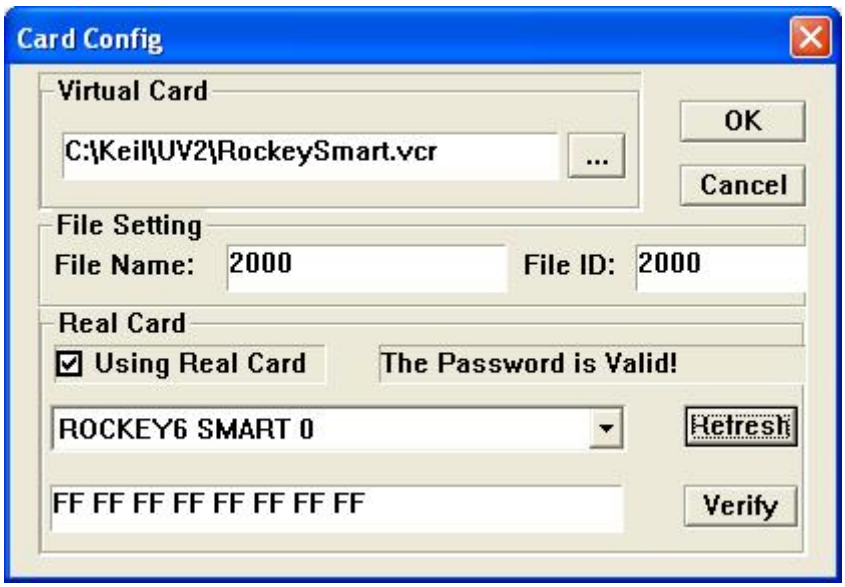


Figure 7-5

Finally, configure the debugger for downloading. There will appear an icon  in the main frame, which can be used for burning the program to the real card after the user debugs it. No matter if burning to a real card or a virtual card, the aim of burning to a virtual card is getting ready for the bulk burn. Details are shown in Figure 7-6.

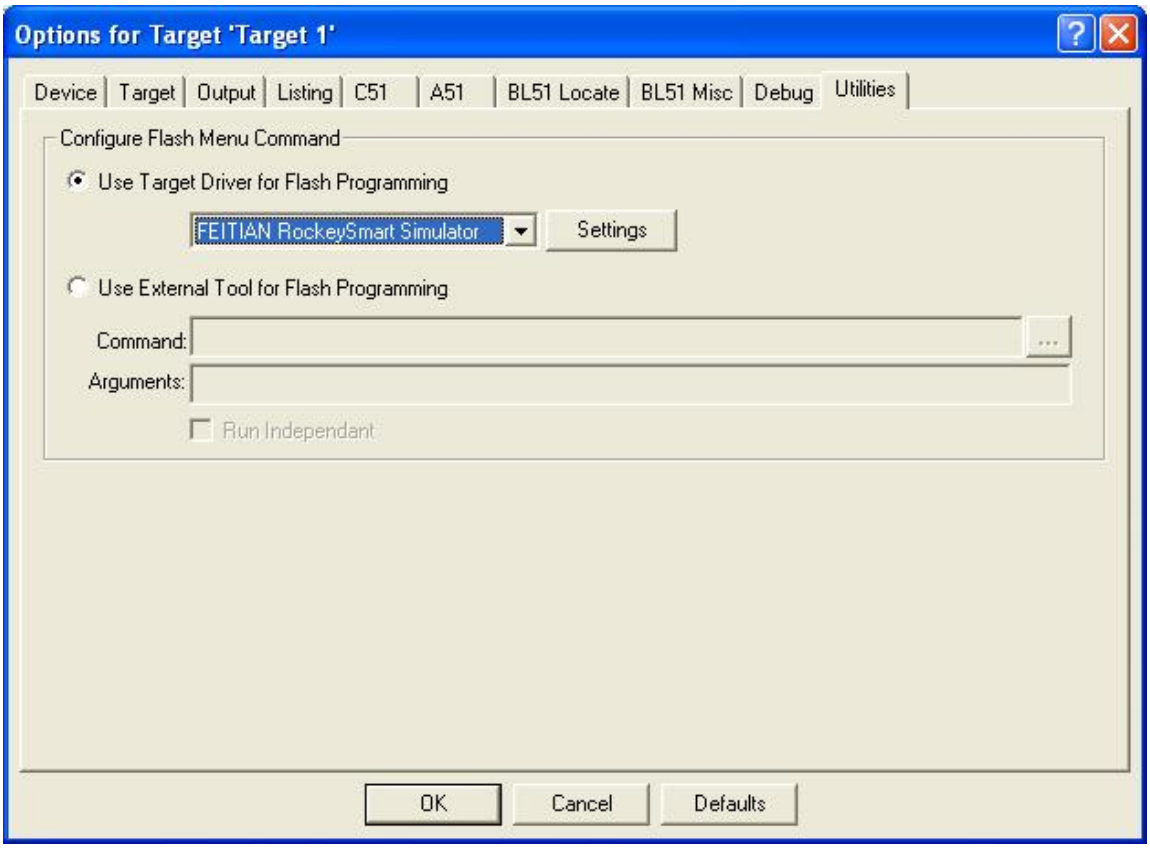


Figure 7-6

Select "View" -> "Project window" spread the tree and right click "source Group1"; select "Add files to Group" "source Group1" to add the corresponding head file and library file in "API\C51" into the project. See Figure 7-7.

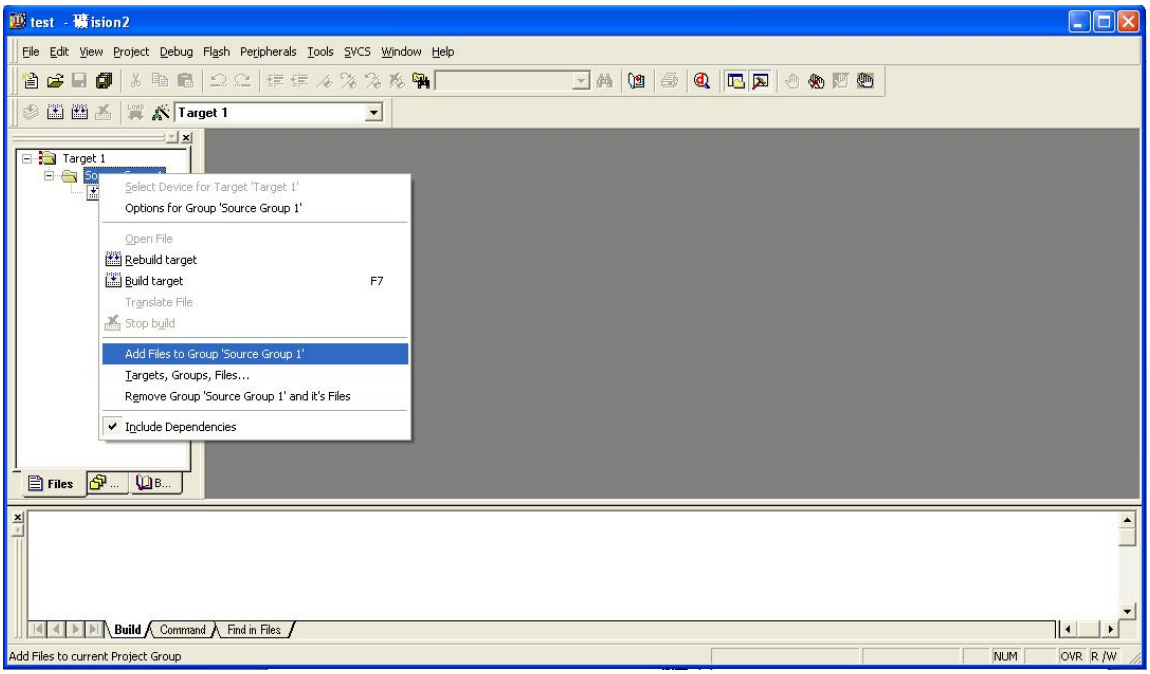


Figure 7-7

When debugging the program, RySSimulator.dll simulates the ROCKEY6 SMART and explains and performs functionality as per the dongle. Therefore the user does not really need a card to perform an operation on an executable file.

After adding "API32\C51\sys_api.h" and "small_mode.LIB" into the project, the user can debug and burn the card. The user does not need to configure the output file. Output is controlled by Keil IDE and the debugger.

7.4 Debugging

After performing the above configuration steps, the program will be in the debugging state. At this point, the Keil uVision2's "simulator" can be used to debug (choose "Use Simulator" button from the upper left part of Figure 2-3). However, there is one difference - the register shows in menu "View > Project Window" the VM51 register rather than the register of "Keil uVision2 Simulator" register. For details of using and debugging "Keil uVision2", please refer to its user manual.

7.5 Exiting

To end the debugging process, "exit()" has to be used at end of the debugging program. ROCKEY6 SMART Simulator will end once it executes "exit ()". It is the same as the user choosing the "Debug > Start/Stop Debug

Session" menu. In other words, if the user chooses the "Debug > Start/Stop Debug Session" menu, the debugging process can be ended at any stage. If the PC pointer refers to address "0", the process will be restarted.

7.6 Sample Testing

ROCKEY6 SMART Simulator will debug and test all examples under the directory "\samples\keil\". The results are exactly the same as the descriptions in the "readme.txt" file.

Note: The simulator cannot simulate the function "sys_recall", because this function is meaningless in the simulator. In addition, please do not use any other methods to generate the "BIN" file into the card for execution. If you want to input the code into the card for execution, please choose "Download" from the menu "Flash", or you can click the tools bar "Download to Flash Memory" button.

7.7 Writing Programs to Real Card

If the "Using Target Driver for Flash Programming" in the Utilities page is set, the program is ready to write into the real card. The only thing that needs to be done is to select "Flash > Download" from "Keil". After that, the whole procedure is finished.

7.8 Summary

In general, ROCKEY6 SMART Simulator not only controls all compilation and execution of the "VM51", but also provides all ROCKEY6 SMART services, such as various file operations, choosing files, reading, writing, creating, float type calculations, and card information operations etc. With these advantages, VM51 developers can easily use the simulator to debug VM51 programs.

Chapter 8. ROCKEY6 SMART Essential

8.1 Development Introduction

ROCKEY6 SMART is a programmable dongle, it has two development models: one is dongle internal program; another is the external program that communicates with the program inside the dongle. After the dongle software and hardware are setup, please follow the steps below to develop a simple protection program.

8.2 Getting Started

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

The following diagram represents the process for dongle configuration:

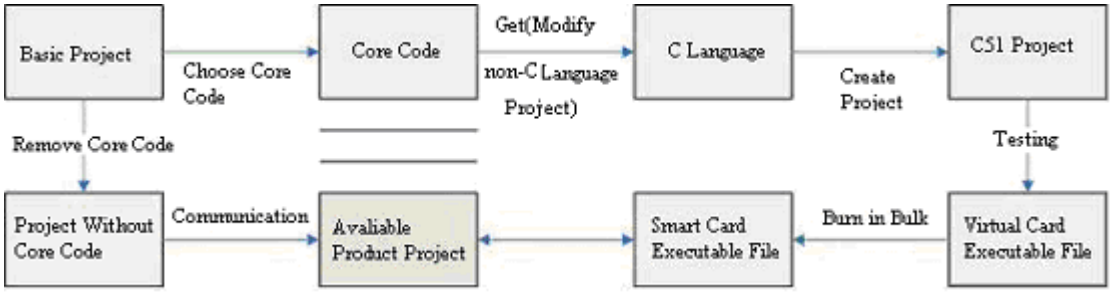


Figure 8-1 Development Process

Please consider the following algorithm: Input 15 bytes ID numbers, its output is 18 bytes numbers. This is a simple message-digest algorithm. Assume these numbers are a part of critical code, and then we save these numbers into the dongle. An example is as follows:

```
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    }
```

The following example is written in C51 C language. It will do the same thing.

```
unsigned char Wi[18] = {7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2,1};
char Ai[11]={'1','0','x','9','8','7','6','5','4','3','2'};
void main(void)
{
int i,j,s;
byte ID[15],newID[18];
s=0;
get_input(ID,0,0,15);//Input original ID card number
memcpy(newID,ID,6);
newID[6]='1';
newID[7]='9';
memcpy(newID+8,ID+6,9);
for(i=0;i<17;i++)
{
j=(newID[i]-48)*Wi[i];
s+=j;
}
s%=11;
newID[17]=Ai[s];
set_response(18, newID);//Outputnew ID card number
exit();//End program
}
```

The 3rd 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 program ending.

In the KEIL compile environment, C language is used. However, it is different with standard C language due to its

distinguished methods for defining variables. For example: "unsigned char xdata buf[128]". "xdata" means putting the variable into the "xdata" area. If a lager array is defined, it is usually assigned at the "xdata" area in order to obtain enough memory space. Please refer to the C51 user manual for further details and features of C51.

8.3 Creating C51 Project

Convert the core code into KEIL C language, then create a C51 project. This part of the core code can 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 this step is completed, you can start to debug, compile and/or download your program.

8.4 Creating Executable File in Dongle

Once the project is created, the source code can be compiled and debugged. It can be debugged using different methods, such as in steps or in blocks. See Figure 8-2:

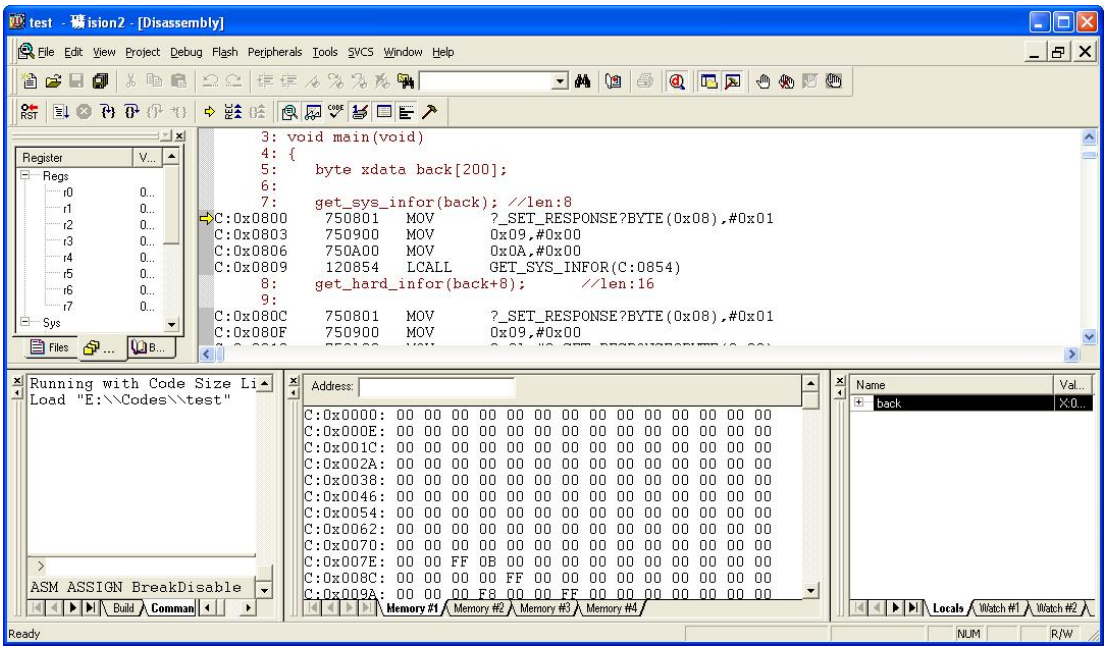


Figure 8-2 Debugging window

If the debugger meets the function "get_input" and/or "set_response", it will display the following window for

data input or data content display. (The example will use "getversion" function).The difference from running the card is when the debugger meets more "get_input" functions, it will display more dialog boxes for user input. In debugging, the user can ignore this parameter. As shown in Figure 8-3.

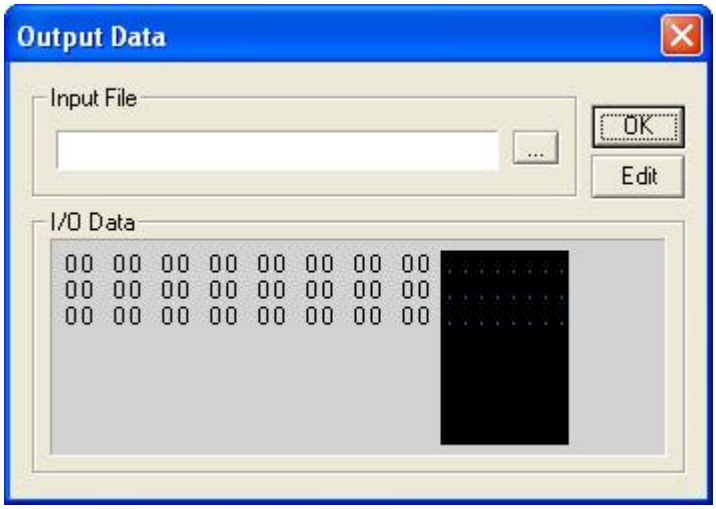


Figure 8-3 Output data

Once the code is successfully compiled, it can be downloaded to the card by using the discussed method in the previous section. Now, if you open the IDE tool to browse real devices, you will find a file with its name and ID being 2000. When the file is executed, it can dynamically fetch the files it needs, and return the corresponding results. As shown in Figure 8-4, Figure 8-5.

File (folder) name	ID	Class	Size	Cl...	Property	Securit...
	2F01	FF	15	00	File sys...	0
TestDir	0001	FF	32	00	Folder	0
1000	0002	FF	40...	00	Exec	0
2000	2000	FF	6065	00	Exec	0

Figure 8-4 Files in the real card

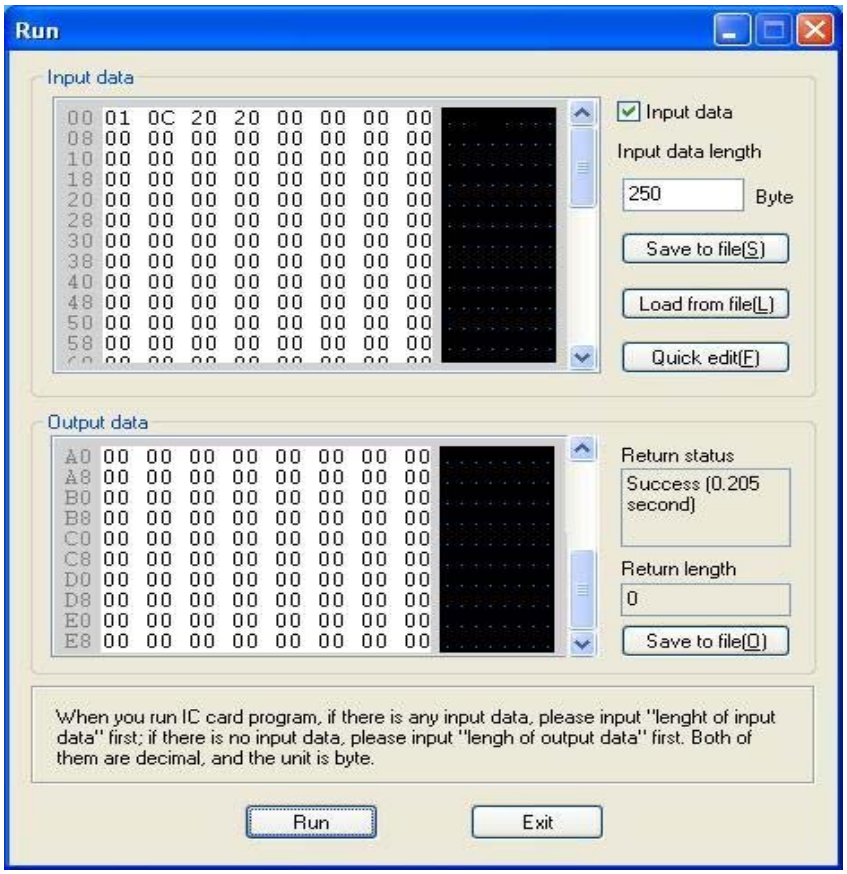


Figure 8-5 Simulated Executions

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

8.5 Editing and Encrypting Dongle Intercommunication Program

The steps for executing a dongle file:

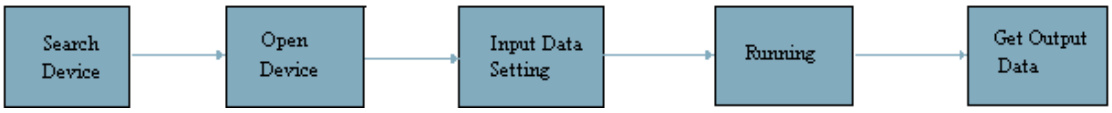


Figure 8-6 File Execution Steps

The following is a configured API that is used for communications between application programs and the ROCKEY6 SMART dongle:

1) EXTERN_C int WINAPI DIC_Find(DWORD UID);

The function is used for dongles attached to the PC, based on the user code, and returns the total numbers of attached dongle devices.

2) EXTERN_C int WINAPI DIC_Open(int hic, char* reader_name);

Open a dongle by its index, and return the handle of the dongle.

3) EXTERN_C int WINAPI DIC_Command(int hic, int cmd, void* cmddata);

Send commands to the dongle. The first parameter is the handle of the dongle; the second parameter is command macro; the third parameter is the inputting/outputting structures according to the command macro.

4) EXTERN_C int WINAPI DIC_Set(void* xdata, int p1, int p2, int p3, char* buffer);

Help the user to put data into "data" in DIC_Command.

5) EXTERN_C int WINAPI DIC_Get(void* xdata, int p1, int p2, char* buffer);

Help the user to take data from "data" of DIC_Command.

Please note that these two functions (DIC_Set and DIC_Get) are not used for any dongle operations. They are simply used to set a buffer. The address of the buffer can be transferred to the 3rd parameter of the function "DIC_Command". This buffer is the required structure. For all structures supported C languages, it is not necessary to use these two functions. According to the definition of the function "DIC_Command", these languages can directly work on the various structures based on different macros. The purpose of these two functions is to help users to configure complicated data structures; especially for those languages which do not support "struct", such as VB etc.

Here is an example of converting the "ConvertID" to the function that can communicate with the 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();//Find the 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;//No card found
14      bD->RunID=0x2000;//Set name of executable file
15      bD->ParaSize=15;//Length of input buffer
16      memcpy(bD->Para,ID,15);//Input buffer
17      int ret=DIC_Command(hic,RUN,bD);//Execute file and wait for data returned
18      memcpy(newID,aD->Result,aD->ResultSize);//Retrieve returned data
19  }
```

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

8.6 Core Code Selection

From the 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 experience, many users are using dongles for only searching if they are attached to the PC, or only storing a piece of dispensable code inside the dongle. Dongles used in this way cannot protect your software well. We recommend users take more time to configure their dongles. Following are some important principles and methods used for dongle applications.

- ✓ Code inside the dongle has to be vital. This part of code has to be saved inside the dongle. Nobody can get to it without undergoing the valid verification procedures and the main program cannot be executed entirely and successfully without this vital part of code. As a result, if hackers attempt to avoid this piece of code the whole program cannot be executed successfully.
- ✓ Try to choose the specific code that relates to your application. Since popular algorithms are well known, software hackers can generally "guess" the functionalities of the code and simulate them to crack the software. Based on the fact that most of our clients are skilled software programmers in their own software applications, we recommend our clients to add their own special code in the dongle. It is impossible for a hacker to know all

the code from various application fields and the dongle containing the special code would be too hard to be broken for a hacker. For example, a graphics programmer could use part of the code from the graphic processing software in the dongle; a PC games programmer could use part of the code from the artificial intelligence algorithms; a mechanical engineer could use part of the code from the mechanics algorithms etc. In this way, the dongles would have dynamic diversity for its security algorithm applications and would make cracking the software too difficult to accomplish.

✓ From the performance aspect, do not give a heavy workload to the dongle. The code inside the dongle can be executed within the dongle. Since the speed of the dongle microprocessor is slower than the PC CPU, the dongle usually creates a bottleneck for the 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 may be slower but won't be crashed.

8.7 Summary

From the above, we have discussed the fundamental methods for using the software protection dongle. Moreover, we have also shown how to separate and convert the code into C language. Finally, we have 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.

Chapter 9. ROCKEY6 SMART Communication API Reference

9.1 int DIC_Find()

Category	Description
Function	Find dongle devices connected to the PC
Input	No
Return value	If return value is less than 0, then it means the return value is incorrect. Please refer to section 9.9 for details. If returned value is 0, it means there is no connected device found. If the returned value is greater than 0, then the returned value is the actual numbers of devices. (NOTE: In most cases, although there are no external devices attached, because device driver is installed in the PC; then it looks to see if the external device exists. To ensure this, it is necessary to use command "DIC_Open" to check if it is true.) Theoretically, the maximum external connected card readers are 32.
Usage Description	In most cases, this command is only used once when the program is initiated by the user. It can result in some initiation work, such as listing all internal card readers. If the user is using concurrent multi programs, then it is only the main program that needs to call this command once. If the user plugs physically in/out the external card dongle, it will not cause any number changes for the displayed card dongle.

9.2 int DIC_FindByMgrCode(void * pMgrCode)

Category	Description
Function	Search encryption devices connected to the computer
Input	Manufacturer management code
Return value	<0 :Error ,Refer to the error code
	=0 :No device
	>0 :Return the number of connected dongles
Usage Description	Normally, this command is used for initializing the dongle and creating the dongle list. We recommend developers invoke this command at the beginning of the main thread.

9.3 int DIC_Open(int hic, char* reader_name)

Category	Description	
Function	Open selected card reader	
Input	hic	(1) Positive value is an enumerate value and is a number of 0, 1, 2 ... which is in the range of returned values of "DIC_Find()". If "reader_name" is not empty, then its return value is the name of the card reader. (2)Negative values, in this case, the parameter of "reader_name" cannot be empty. "DIC_Open" will try to directly use the reader name from the "reader_name" to open the corresponded card reader. If the card reader is a virtual device, then it is the path of the file.
	reader_name	It is a parameter of input/output, please see also "hic" description
Return value	> = 0	Represents success. The returned value is the opened handler.
	< 0	The return value is an error code; please refer to section 9.9 for details.
Usage Description	It results in establishing the unique connection to the connected device. Once the operations are finished, the user should release the resources held by the connection via command "DIC_Close". It is more important for using concurrently multi programs.	

9.4 int DIC_Close(int hic)

Category	Description
Function	Close selected card reader
Input	Card handler, it will be ok if it is the same as the "DIC_Open" returned "hic".

Return	The return value is an error code; please refer to section 9.9 for details.
Usage Description	It is used for releasing the connection between the PC and external devices. At the same time, clear the data and security state in the device memory.

9.5 int DIC_Command(int hic， int cmd， void* data)

Category	Description	
Function	Finish the operation on selected device	
Input	hic	Device handle
	cmd	Specific operation; it usually uses various constants to represent different operations. Please also see [cmd Description] for details.
	data	All command related input/output data, please also see [special data structure] description
Return	The return value is an error code; please refer to section 9.9 for details.	
Usage Description	It is a multi-functional command, all devices related to this operation are accomplished by this command	

[cmd Description]

Every command constant is already defined as a macro in "Dic32u.h", referring to different operations. Some of the macros may conflict with macros predefined under various integrated development environments, and pre-compiling will not report such conflicts. If the syntax convention is correct but cannot perform the corresponding functions, please take the relevant constants in "Dic32u.h" as parameters or rename the related macros before they are used. Normally, there is no need to care about the related data structures, use mode operation to get and set the data structures. Detailed descriptions are available under "Dic_Det()" and "Dic_Set()".

Macro	Data Structure	Operations
GET_CARD_INFO	DICST_CardInfo	Get smart card volume and manufacturer information
GET_HARDWARE_INFO	DICST_HardInfo	Get hardware manufacture date, serial number, shipping time and COS version
GET_MANAGER_CODE	DICST_ManagerCode	Get management code; include

		zone code, reseller code and two user's codes.
GET_CARD_PRIVILEGE	1 byte	Get card current password verification state. There are 3 returned values: 0(password not verified), 4(super password verified), 8(remote update password verified)
GET_REMOTE_INFO	DICST_RemoteInfo	Get remote update information
SET_REMOTE_INFO	DICST_RemoteInfo	Setup remote update information
CHECK_REMOTE_INFO	DICST_RemoteInfo	Verify remote update information
GET_CURRENT_DIR	DICST_Dir	Get current directory information
SET_CURRENT_DIR	DICST_Dir	Setup current directory information
GET_CURRENT_FILE	DICST_File	Get current file information
SET_CURRENT_FILE	DICST_File	Setup current file information
GET_PARENT_DIR	DICST_Dir	Get upper directory
SET_PARENT_DIR	DICST_Dir	Setup upper directory
LIST_DIR	Input 1byte index, specifying the file to be opened, index: 0-255. DICST_Dir or DICST_File	List directory structure of file system; it is determined by the "FILEATTR_DIR" from the returned data.
READ_FILE	DICST_Before_Read_Data and DICST_After_Read_Data	Read current file (first read file information, then its content)
WRITE_FILE	DICST_Write_Data	Write a file into the card
FORMAT_CARD	DICST_System_Info	Format the card
CREATE_DIR	DICST_Dir	Create a directory
CREATE_FILE	DICST_File	Create a file
REMOVE_DIR	DICST_Dir	Delete a directory
REMOVE_FILE	DICST_File	Delete a file
RANDOM	char array	Get a random number
RUN	DICST_Before_Read_Data and DICST_After_Run_Data	Run an executable file
CRYPTOTEXT_FILE	none	Encrypt a file
PLAINTEXT_FILE	none	Decrypt a file
CHECK_SUPER_PASS	char data[]	Verify the super password
SET_SUPER_PASS	DICST_SuperPass_Data	Setup the super password
GET_UPGRADE_REMOTE_PASS	DICST_Upgrade_RemotePass	Get remote update information
DESENC	DICST_Des_Data and DICST_After_EncDec_Data	DES encryption
DESDEC	DICST_Des_Data and DICST_After_EncDec_Data	DES decryption
RSAGENKEY	DICST_Rsa_GenKey	RSA generates key pairs
RSAENC	DICST_Rsa_Data and	RSA encryption

	DICST_After_EncDec_Data	
RSADEC	DICST_Rsa_Data and DICST_After_EncDec_Data	RSA decryption
GETFREESPACE	Returned value is a DWORD	Get the remaining space of the file system
SETCOUNTER	Inputted value is a DWORD	Begin the maximum using counter
STEPCOUNTER	none	reduce the using counter numbers in each step

[Special Data Structure]

All operations of data structures can be accomplished with mode operations in "DIC_Get" and "DIC_Set". Normally, there is no need to care about these data structures, it is not recommended to use the structures, because it would reduce their transferability.

```
============================================================
DICST_CardInfo      /*Specified by developer*/
typedef struct{
      char volume[16];       //Volume label of IC card
      char atr[15];          //Developer information
} DICST_CardInfo;
============================================================
DICST_HardInfo      /*Information set by Feitian, software developer is not authorized to modify it */
typedef struct{
      DWORD FactoryTime;    //Production date and time
      DWORD HardSerial;     //Serial number
      DWORD ShipTime;       //Shipping date and time
      DWORD COSVersion;     //COS version
} DICST_HardInfo;
============================================================
DICST_ManagerCode     /*Information set by Feitian, software developer is not authorized to modify it */
typedef struct{
      WORD Zone;       //Region code
      WORD Agent;      //Agent code
      WORD User1;      //User code 1
      WORD User2;      //User code 2
} DICST_ManagerCode;
============================================================
DICST_RemoteInfo  /*Specified and explained by developer*/
typedef struct{
      DWORD RemoteTag;      //Remote update flag
      BYTE RemotePass[8];     //Remote update password
} DICST_RemoteInfo;
============================================================
```

```
DICST_Dir
typedef struct {
    WORD        dirid;          // Directory ID
    BYTE        dircla;         // Directory class
    BYTE        diratrpri;      // Directory properties and security level
    WORD        dirsize;        // Unused
    char        dirname[16];      // Directory name
} DICST_Dir;
============================================================
DICST_File
typedef struct {
    WORD        fileid;         // File ID
    BYTE        filecla;        // File class
    BYTE        fileatrpri;     // File properties and security level
    WORD        filesize;       // File size
    char        filename[17];   // Filename
} DICST_File;
============================================================
```

Definitions of diratrpri(directory attribute) and fileatrpri(file attribute):

Label	Type
FILEATTR_NORMAL	Normal data file
FILEATTR_EXEC	Executable
FILEATTR_DIR	Directory
FILEATTR_UPIGNORE	If the current security level is already higher than the executable program, the program cannot be executed.
FILEATTR_INTERNAL	File used by internal program and cannot be operated externally. If an executable file is marked as internal, it has hiding properties, such files can be viewed when listing directory only if the super password is verified.

```
============================================================
DICST_Upgrade_RemotePass /*Used in remote update management*/
typedef struct{
    DWORD RemoteTag;    //Remote update flag
```

```
    DWORD HardSerial;     //Hardware serial number
    BYTE RemotePass[8];   //Remote update password
} DICST_Upgrade_RemotePass;
============================================================
DICST_Before_Read_Data
typedef struct{
    WORD        offset;
    WORD        size;
} DICST_Before_Read_Data;
============================================================
DICST_After_Read_Data
typedef struct{
    WORD        readedsize;
    char        buffer[1];          // The size equals to the value of "readsize"
} DICST_After_Read_Data;
============================================================
DICST_Write_Data
typedef struct{
    WORD        offset;     //File offset
    WORD        size;       //File size
    char        buffer[1];      // The size equals to the value of "size"
} DICST_Write_Data;
============================================================
DICST_System_Info   /*The volume label and developer information of the card must be specified after formatting*/
typedef struct{
    char    volume[16];     //The volume label
    char    atr[15];        //The developer information
} DICST_System_Info;
============================================================
DICST_Before_Run_Data
typedef struct{
    WORD RunID;             //File ID
    WORD ParaSize;          //Parameter size
    BYTE Para[1];           // The size equals to the value of "ParaSize"
} DICST_Before_Run_Data;
============================================================
DICST_After_Run_Data
typedef struct   {
    WORD ResultSize;        //The number of bytes of the parameter is returned
    BYTE Result[1];         // The size equals to the value of "ResultSize"
} DICST_After_Run_Data;
============================================================
DICST_SuperPass_Data
```

```
typedef struct {
        BYTE MaxTryTimes;         //The maximum number of retries allowed; cannot be 0
        BYTE SuperPass[8];          //Super password; cannot be all zeros (leading to locking)
} DICST_SuperPass_Data;
============================================================
```

9.6 int DIC_Get(void* target， int p1， int p2， char* pstr)

Category	Description	
Function	Get data	
Input	target	Source data buffer
	p1	Mode/shift, if the highest bit is 1, then it is a mode operation; otherwise, it is a user defined shift value
	p2	Mode/shift, if the highest bit is 1, then it is a mode operation; otherwise, it is a user defined shift value
	pstr	Return mode / size, the highest 2 bits represent return mode, others indicate the operation size
Return	If return mode is BY_VALUE, the return data is the data required by the user. If it is BY_ARRAY, the returned value is the data length of pstr buffer. If it is BY_STRING, the returned value is the character string length of pstr buffer. Note: For some model operations like vendor information, no matter what return mode is set, it will return from pstr. Then if pstr is set to NULL, the return of -1 indicates that the required operations can not be performed	
Usage Description	The data pointer in DIC_Command can be regarded as a structure pointer in C. But for many languages that do not support 'structure", DIC_Get and DIC_Set are for storing to / getting from the 'structure"	

[p1 description]

For the p1 parameter of DIC_Get and DIC_Set, some macros are defined to specify the operation objects. The highest bit of the macros is set to 1. The following are the pre-defined macros:

Macro	Description
FILL	Fill
ATR	Manufacturer information
VOLUME	Volume
FACTORY_TIME	Manufacture date
HARD_SERIAL	Hardware serial number
SHIP_TIME	Shipping time
COS_VERSION	COS version
ZONE	Zone code
AGENT	Reseller code
USER1	User code 1
USER2	User code 2
DIR_ID	Directory ID
DIR_CLASS	Directory category
DIR_ATTRIBUTE	Directory attribute
DIR_PRIVILEGE	Directory privilege status
FILE_NO	File number
DIR_NAME	Directory name
FILE_ID	File ID
FILE_CLASS	File category
FILE_ATTRIBUTE	File attribute
FILE_PRIVILEGE	File security level
FILE_SIZE	File size
FILE_NAME	File name
WRITE_DATA	Write data

READ_DATA	Read data
RUN_DATA	Runtime input/output data
RANDOM_SIZE	Random number size (in byte)
REMOTE_TAG	Remote update tag
REMOTE_PASS	Remote update password
UPGRADE_HARD_SERIAL	Hardware serial number
UPGRADE_REMOTE_TAG	New remote update tag
UPGRADE_REMOTE_PASS	New remote update password
REMOTEPASS_STATUS	Remote update status
SUPERPASS_STATUS	Super password verification status
SUPERPASS_DATA	Super password
AFTER_ENCDEC_DATA	Get encrypted/decrypted data
DES_KEYFILEID	ID of DES key file
DES_KEYINDEX	DES key ID
DES_DATA	Data to be encrypted (DES)
RSA_DATA	Data to be encrypted (RSA)
RSA_PUBKEYSIZE	RSA public key ID
RSA_PIVKEYID	RSA private key ID
RSA_PUBKEYSIZE	Length of RSA key

[p2 Description]

For the p2 parameter of DIC_Get and DIC_Set, the highest 2 bits indicate data transfer or return mode, the default value of the lower 30-bit is 0, if not, it refers to the operation byte number.

Macro	Description
BY_VALUE	Value transfer
BY_ARRAY	Array transfer
BY_STRING	In term of character

[Example]

```
// If the highest bit of p1 is not 1, then the function is to get the user data.
char tstr[] = "Hello World!";
char pstr[20];
```

```
DIC_Get(tstr， 3， BY_ARRAY | 5， pstr);
```

Then, the content of pstr is "lo Wo", 3 is the shift value, and 5 is the size.

9.7 int DIC_Set(void* target， int p1， int p2， int p3， char* pstr)

Category	Description	
Function	Data setup	
Input	target	Target buffer zone
	p1	Mode/shift, if the highest bit is 1, it is a mode operation, otherwise, it is user defined shift value
	p2	Transfer mode/ size, the highest 2 bits specify the transfer mode, others indicate the operation size
	p3	Normally, it is the user value to be stored
	pstr	Character type buffer of the source data, it can be NULL
Return	0 means a success, otherwise is –1.	

[Example]

```
DIC_Set(data， FILL， 256， 0， NULL); // Clear the 256-byte data

DIC_Set(data，FILE_ID， BY_VALUE， 0x2100， NULL);// Set the file ID in the data

DIC_Set(data ，RUN_ID,BY_VALUE ，0xA1B2 ，NULL);// Set the file ID to be executed in the data
```

9.8 int DIC_GetVersion(char* ver)

Category	Description
Function	Get the dynamic library version
Input	"ver" is used to return the version number of the dynamic library. The content is: [V/R]xx.yy For example: R01.10 means it is the "real card dynamic library version 1.10"

	Another example: "R03.02" means it is "virtual card dynamic library version 3.02".
Return	An error code is returned, please see also section 9.9.

9.9 Returned Error Codes

Code (hex)	Code (decimal)	Meaning
0x00000000	0	Correct, no error.
0x80100001	-2146435071	Internal connection check failed
0x80100002	-2146435070	Operation is terminated by the user
0x80100003	-2146435069	Incorrect operation handler
0x80100004	-2146435068	Incorrect parameters
0x80100005	-2146435067	Invalid registering information or the information is lost
0x80100006	-2146435066	No enough memory to accomplish the instruction
0x80100007	-2146435065	Internal overtime
0x80100008	-2146435064	The user defined buffer is too small to hold all returned data
0x80100009	-2146435063	Card reader is unknown
0x8010000A	-2146435062	User assigned time exceeds
0x8010000B	-2146435061	The card is occupied by the other connection
0x8010000C	-2146435060	There is no card in the card reader
0x8010000D	-2146435059	Unknown card type
0x8010000E	-2146435058	Card reader is unable to quit card operation
0x8010000F	-2146435057	The current card does not support the user defined communication protocols
0x80100010	-2146435056	Card is not ready for receiving instructions
0x80100011	-2146435055	Some variable values are invalid

0x80100012	-2146435054	Operations are terminated by the system. You may need to re-logon or restart the machine.
0x80100013	-2146435053	Internal communication error
0x80100014	-2146435052	Unknown internal error
0x80100015	-2146435051	Invalid manufacturer information
0x80100016	-2146435050	User has tried to stop a process that it is not actual existing.
0x80100017	-2146435049	The selected card reader is not available right now.
0x80100018	-2146435048	Operation is stopped; the service programs could have quit.
0x80100019	-2146435047	The receiving buffer of PCI is too small
0x8010001A	-2146435046	The driver of card reader does not support current card reader.
0x8010001B	-2146435045	The card reader driver cannot get the unique name, due to a duplicated name.
0x8010001C	-2146435044	Card is not supported by current card reader.
0x8010001D	-2146435043	The smart card services do not start.
0x8010001E	-2146435042	The smart card services are stopped.
0x8010001F	-2146435041	An unexpected card error occurs.
0x80100020	-2146435040	Unable to get provider's information of the smart card.
0x80100021	-2146435039	Unable to get manufacturer's information of the smart card.
0x80100022	-2146435038	The user requested functions are not supported by current smart card.
0x80100023	-2146435037	The selected directory does not exist.
0x80100024	-2146435036	The selected file does not exist.
0x80100025	-2146435035	The selected directory is invalid.

0x80100026	-2146435034	Selected file is invalid. No file has been selected now.
0x80100027	-2146435033	Access denied for current file
0x80100028	-2146435032	The space of card is full. No information could be written in.
0x80100029	-2146435031	Configuration file pointer error
0x8010002A	-2146435030	PIN code error
0x8010002B	-2146435029	An unidentified error code returned from the smart card service.
0x8010002C	-2146435028	Requested certificate does not exist
0x8010002D	-2146435027	The request for getting the certificate is denied.
0x8010002E	-2146435026	Cannot find any card reader
0x8010002F	-2146435025	Data is lost during the process of the smart card communication, please try again.
0x80100030	-2146435024	Requested key file does not exist
0x80100065	-2146434971	Card reader cannot communicate with card due to configuration conflicts of manufacturer information
0x80100066	-2146434970	Card has no response for reset operation
0x80100067	-2146434969	Card is power off
0x80100068	-2146434968	The card is repositioned. The shared information is invalid.
0x80100069	-2146434967	The card is disconnected.
0x8010006A	-2146434966	Access denied due to existing security settings.
0x8010006B	-2146434965	PIN code is not verified, access denied.
0x8010006C	-2146434964	The maximum numbers of PIN verifications are reached. Access denied.
0x8010006D	-2146434963	The last card file has been reached. There is no more files could be visited.

0x8010006E	-2146434962	Operation is stopped by the user.
0x8010006F	-2146434961	The smart card PIN is not configured.
0xA0100001	-1609564159	The file already exists.
0xA0100002	-1609564158	Card internal storage operation error.
0xA0100003	-1609564157	The user provides an invalid CLA.
0xA0100004	-1609564156	The user provides an invalid INS.
0xA0100005	-1609564155	The virtual machine address is overflow/ normal
0xA0100006	-1609564154	Divide by zero error
0xA0100007	-1609564153	The card is not plugged into the correct position.
0xA0100008	-1609564152	The card is in unknown state.
0xA0100009	-1609564151	The card is not opened.
0xA010000A	-1609564150	Unknown command
0xA010000B	-1609564149	The reset times of setting super password is "0"
0xA010000C	-1609564148	Opened too many devices.
0xA010000D	-1609564147	Invalid instruction error.
0xA01000FF	-1609563905	The card still has data needing to be returned.
0xF0100001	-267386879	Unable to find the dongle.
0xF0100002	-267386878	No valid dongle available.
0xF0100003	-267386877	Error writing when opening the device.
0xF0100004	-267386876	Error reading when opening the device.
Virtual Device Error Codes		
0xA0101001	-1609560063	Failed to create virtual card file
0xA0101002	-1609560062	Failed to open virtual card file

9.10 Permissions

1. Permission Types

0	Password not verified.
1	Agent password verified.
2	Manufacturer password verified.
3	Remote update password verified.
4	Super password verified.

Required permission levels for various operations:

GET_CARD_INFO	0, 1, 2, 3, 4
GET_HARDWARE_INFO	0, 1, 2, 3, 4
GET_MANAGER_CODE	0, 1, 2, 3, 4
GET_CARD_PRIVILEGE	0, 1, 2, 3, 4
GET_REMOTE_INFO	0, 1, 2, 3, 4
SET_REMOTE_INFO	0, 1, 2, 3, 4
CHECK_REMOTE_INFO	0, 1, 2, 3, 4
GET_CURRENT_DIR	0, 1, 2, 3, 4
SET_CURRENT_DIR	0, 1, 2, 3, 4
GET_CURRENT_FILE	0, 1, 2, 3, 4
SET_CURRENT_FILE	0, 1, 2, 3, 4
GET_PARENT_DIR	0, 1, 2, 3, 4
SET_PARENT_DIR	0, 1, 2, 3, 4
LIST_DIR	0, 4, (4 could list hidden executable files)
READ_FILE	0, 1, 2, 3, 4
WRITE_FILE	0, 1, 2, 3, 4 (executable file, internal used only files are required level 4)
FORMAT_CARD	1, 2, 4
CREATE_DIR	0, 1, 2, 3, 4
CREAT_FILE	0, 1, 2, 3, 4 (executable files, internal use only file requires level 4)
REMOVE_DIR	0, 1, 2, 3, 4 (required DIR is empty)
REMOVE_FILE	0, 1, 2, 3, 4 (executable files, internal use only file requires level 4)
RANDOM	0, 1, 2, 3, 4
RUN	0, 1, 2, 3
CHECK_SUPER_PASS	0, 1, 2, 3, 4

SET_SUPER_PASS	4
GET_UPGRADE_REMOTE_PASS	4
CHECK_AGENT_PASS	0, 1, 2, 3, 4
CHECK_MANUFACTURER_PASS	0, 1, 2, 3, 4
SET_HARDWARE_INFO	2
SET_MANAGER_CODE	2
SET_AGENT_PASS	2
SET_MANUFACTURER_PASS	2
DESENC	0, 1, 2, 3, 4
DESDEC	0, 1, 2, 3, 4
RSAGENKEY	4
RSAENC	0, 1, 2, 3, 4
RSADEC	0, 1, 2, 3, 4
GETFREESPACE	0, 1, 2, 3, 4
SETCOUNTER	4
STEPCOUNTER	0, 1, 2, 3, 4

Chapter 10. API Reference And Samples

10.1 API Reference

The API is a built-in interface that is used for the program communications between the developer's applications and the ROCKEY6 SMART dongle. It is provided to the developer by using a corresponding develops language. For Microsoft Visual C++ developers, there are two methods to use them: "static link" and "dynamic link". When 'static link" is used, you need to add the file "Dic32u.lib" and "Dic32.h" into the project.

There are a total of 8 API interfaces in "Dic32u.DLL". The following table provides a description (please also see Chapter9)

Table 10.1-1 API Introduction

ROCKEY6 SMART API Interface	Description
DIC_Find	Find ROCKEY6 SMART devices connected to computer
DIC_FindByMgrCode	Find ROCKEY6 SMART devices connected to computer according to the management code
DIC_Open	Open selected card reader
DIC_Close	Close selected card reader
DIC_Command	Dongle operation instruction function is the main function for sending the instructions to the dongle
DIC_Get	Get returned data sub function
DIC_Set	Set inputted data sub function
DIC_GetVersion	Get current version number of dynamic library

10.2 Sample01: Fundamental Framework

"DIC_Find", "DIC_Open" and "DIC_Close" are the three basic API operations for find open and close the dongle. DIC_Find returns the number of the smart card reader drivers installed. Please note that even though there is no ROCKEY6 SMART dongle connected to the PC, the return value for "DIC_Find" is NOT zero, for its driver is already installed. To identify whether a ROCKEY6 SMART is plugged into the PC, "DIC_Open" needs to be run. The

parameters of "DIC_Open" and "DIC_Close" are the dongle handlers that are counted from 0. Once a dongle is unlocked, the root directory is shown.

The following example is using the 'static link" to connect "DLL". Its corresponding project is<Samples\API32VC\VC6\Sample01>.

```
num = DIC_Find();//Number of dongles
if (num <= 0)
{
printf("No dongle found \n");
      return;
}
printf("Open the Dongle \n");
for (i=0;i<num;i++)
{
            hic = DIC_Open(i, NULL);// Do not have to know the name of the dongle,
            if (hic >= 0) break;          // Stop searching once one dongle is found.
}
if (i == num)
{
      printf("No dongle with connected Dongle found\n");
      return;
}
 if (errcode != SCARD_S_SUCCESS)
{
printf("Error code: %08x\n", errcode);
      return;
}
// ...
// Specific dongle operations
// ...
printf("Close the Dongle \n");
errcode = DIC_Close(hic);
if (errcode != SCARD_S_SUCCESS)
{
      printf("Error code: %08x\n",errcode);
      return;
}
```

10.3 Sample02: Traversing Dongles

If a PC has many USB ports or connects to a USB HUB, then more than one ROCKEY6 SMART dongle could be present. More importantly, those dongles could be simultaneously used on the same PC without any conflicts. The maximum numbers of concurrent dongles is 16. The software developer must determine how to distinguish between multiple attached ROCKEY6 SMART dongles. We would recommend the developer to use the dongle management ID information to do this, as this ID is always unique for each customer account. (Unless the customer requests a different ID)

"DIC_Find" is used to find the number of card reader drivers on the system. A successful operation of "DIC_Open" shows the existence of a dongle. A complete example project is in < Samples\API32VC\VC6\Sample02>.

```
char reader_name[256]; // The name of the dongle
num = DIC_Find();
printf("%d\n", num); // Output should be "6", if drivers installed properly
if (num <= 0)
{
	printf("No dongle found \n");
	return;
}

// Traverse begins
for (i=0;i<num;i++)
{
printf("Open the Dongle: ");
	hic = DIC_Open(i, reader_name);
	if (hic < 0)
	{
		if (hic == SCARD_W_REMOVED_CARD)
		{
			printf("No Dongle found \n");
			continue;
		}
		printf("Error code: %08x\n", hic);
		return;
	}
	printf("%s\n", reader_name); // Output the name of the dongle
	// ...
	// Specific Dongle operations
	// ...
```

```
    printf("Close the Dongle %d\n", i);
    errcode = DIC_Close(i);
    if (errcode != SCARD_S_SUCCESS)
{
        printf("Error code: %08x\n", errcode);
        return;
    }
}
```

10.4 Sample03: Dynamic Linking Modes

Loading "DLL" is an optional link mode that is unnecessary to use "lib" files. The complete example project is in <Samples\API32VC\VC6\Sample03>.

```
// Define functions to be referenced
typedef int (WINAPI *fDIC_Find)();
typedef int (WINAPI *fDIC_Open)(int hic, char* reader_name);
typedef int (WINAPI *fDIC_Close)(int hic);
typedef int (WINAPI *fDIC_GetVersion)(char* ver);
typedef int (WINAPI *fDIC_Command)(int hic, int cmd, void* cmddata);
typedef int (WINAPI *fDIC_Get)(void* xdata, int p1, int p2, char* buffer);
typedef int (WINAPI *fDIC_Set)(void* xdata, int p1, int p2, int p3, char* buffer);

// Define functions
HINSTANCE hDll = NULL;        // Dynamic library handle
fDIC_Find       dDIC_Find;
fDIC_Open           dDIC_Open;
fDIC_Close          dDIC_Close;
fDIC_GetVersion     dDIC_GetVersion;
fDIC_Command        dDIC_Command;
fDIC_Get        dDIC_Get;
fDIC_Set        dDIC_Set;

void main()
{
    int errcode, num, hic, i;

    // Load dynamic library
    hDll = LoadLibrary("dic32u.dll");
    if (hDll == NULL) return;
    dDIC_Find = (fDIC_Find)GetProcAddress(hDll, "DIC_Find");
    dDIC_Open = (fDIC_Open)GetProcAddress(hDll, "DIC_Open");
```

```
    dDIC_Close = (fDIC_Close)GetProcAddress(hDll, "DIC_Close");
    dDIC_GetVersion= (fDIC_GetVersion)GetProcAddress(hDll, "DIC_GetVersion");
    dDIC_Command = (fDIC_Command)GetProcAddress(hDll, "DIC_Command");
    dDIC_Get = (fDIC_Get)GetProcAddress(hDll, "DIC_Get");
    dDIC_Set = (fDIC_Set)GetProcAddress(hDll, "DIC_Set");

    printf("Find dongle: ");
    num = dDIC_Find();
    printf("%d\n",num);
    if (num <= 0)
    {
        printf("No dongle found\n");
        return;
    }

    printf("Open dongle\n");
    for (i=0;i<num;i++)
    {
        hic = dDIC_Open(i, NULL);      // Dont need to know the name of dongle, set NULL
        if (hic >= 0) break;           // Stop the searching once one is found
    }
    if (i == num)
    {
        printf("No dongle connected to Dongle found \n");
        return;
    }
    // ...
    // Specific dongle operations
    // ...
    printf("Close the dongle \n");
    errcode = dDIC_Close(hic);

    // Unload dynamic library
    FreeLibrary(hDll);
}
```

10.5 Sample04: Get Developer and Volume Label Information

Software developers specify both volume and software developer information. Developers also decide how to use such information. In general we recommend that developers write their company information into the software developer information file. In doing so after a "DIC_Open" command, you may get this information and use it to

identify your dongle.

"DIC_Command" is the core API through which most of the functions works. It has three parameters: dongle handle, the command to be executed, and data buffer related to said command. The second parameter is already defined as a macro for ease of use. For example, macro "GET_CARD_INFO" is used to read software developer and volume information.

"DIC_Get" is an auxiliary API responsible for helping the user obtains information from the data buffer. Its second parameter is also pre-defined as a macro. If the second parameter is "ATR", it refers to software developer information. If the second parameter is "VOLUME", it refers to volume information. The third parameter defines the return mode of the data. "BY_VALUE" means the information is returned as the return value of "DIC_Get". "BY_ARRAY" means the information returned is stored in the buffer, and the return value of "DIC_Get" is the content length of the buffer. When using "DIC_Get", if the macro is a piece of string information of 'software developer information" or "volume", the data is always returned by "BY_ARRAY", no matter how the returned mode is set. Then, the return value of "DIC_Get" is the length of software developer or volume information in the buffer. Please refer to < Samples\API32VC\VC6\Sample04>.

```
int i, num, hic, errcode;
char cmddata[256];// command buffer
char buffer[256];    // user buffer

num = DIC_Find();
for (i=0;i<num;i++)
{
      hic = DIC_Open(i, NULL);
      if (hic >= 0) break;          // Stop searching once one is found
}
if (i == num) return;

errcode = DIC_Command(hic, GET_CARD_INFO, cmddata);
DIC_Get(cmddata, ATR, BY_ARRAY, buffer);   // Transfer ATR from cmddata to buffer
printf("Developer Info: %s\n", buffer);
DIC_Get(cmddata, VOLUME, BY_ARRAY, buffer); // Transfer volume information from cmddata to buffer
printf("Main Volume Label: %s\n", buffer);

errcode = DIC_Close(hic);
```

10.6 Sample05: Get Hardware Information

Hardware information includes manufacture date and time, hardware serial number, ship date and time and COS version. Such information is available to the macro of "GET_HARDWARE_INFO" in "DIC_Command". And the corresponding macros in "DIC_Get" can also get the information items: "FACTORY_TIME" (The time that the dongle was produced by the manufacturer.), "HARD_SERIAL" (The hardware serial number is a globally unique identifier), 'SHIP_TIME" (The ship time may be related to the software developer warranty), and "COS_VERSION" (COS version of the card).

The following example demonstrates another mode of the "DIC_Get" application. If the return mode is "BY_VALUE", the last parameter can be set to "NULL". You may get the parameter from the return value. Please refer to < Samples\API32VC\VC6\Sample05>.

```
int i, num, hic, errcode;
char cmddata[256];// Command buffer
DWORD FactoryTime, HardSerial, ShipTime, COSVersion;

num = DIC_Find();
for (i=0;i<num;i++)
{
	hic = DIC_Open(i, NULL);
	if (hic >= 0) break;		// Stop searching once one is found
}
if (i == num) return;

errcode = DIC_Command(hic, GET_HARDWARE_INFO, cmddata);
FactoryTime = DIC_Get(cmddata, FACTORY_TIME, BY_VALUE, NULL);
HardSerial = DIC_Get(cmddata, HARD_SERIAL, BY_VALUE, NULL);
ShipTime = DIC_Get(cmddata, SHIP_TIME, BY_VALUE, NULL);
COSVersion = DIC_Get(cmddata, COS_VERSION, BY_VALUE, NULL);
printf("Manufacture Date and Time:	%08X\n", FactoryTime);
printf("Hardware Serial Number: %08X\n", HardSerial);
printf("Shipping Date and Time:	%08X\n", ShipTime);
printf("COS Version:	%c%c.%c%c\n", COSVersion>>24, COSVersion>>16, COSVersion>>8, COSVersion);

errcode = DIC_Close(hic);
```

10.7 Sample06: Get Region Code, Agent Code, User Code 1, and User Code 2

The macro "GET_MANAGER_CODE" of DIC_Command may get the management code information of the dongle. "ZONE", "AGENT", "USER1" and "USER2" of "DIC_Get" represent respectively region code (country code of dongle), agent code (sales agent code of the dongle), and user code (software developers have their own unique codes). Please refer to < Samples\API32VC \VC6\Sample06>.

```
int i, num, hic, errcode;
char cmddata[256];// Command buffer
WORD Zone, Agent, User1, User2;

num = DIC_Find();
for (i=0;i<num;i++)
{
	hic = DIC_Open(i, NULL);
	if (hic >= 0) break;		// Stop searching once one is found
}
if (i == num) return;

errcode = DIC_Command(hic, GET_MANAGER_CODE, cmddata);
Zone = DIC_Get(cmddata, ZONE, BY_VALUE, NULL);
Agent = DIC_Get(cmddata, AGENT, BY_VALUE, NULL);
User1 = DIC_Get(cmddata, USER1, BY_VALUE, NULL);
User2 = DIC_Get(cmddata, USER2, BY_VALUE, NULL);
printf("Region Code:	%04x\n", Zone);
printf("Agent Code: %04x\n", Agent);
printf("User Code 1:	%04x\n", User1);
printf("User Code 2:	%04x\n", User2);

errcode = DIC_Close(hic);
```

10.8 Sample07: Random Number

Random number generation is very important for many encryption algorithms. ROCKEY6 SMART performs random number generation in the hardware. First of all, the length of the random number has to be set through "DIC_Set" (maximum length of 16 bytes, set with "RANDOM_SIZE" macro), and then calls the command

"RANDOM" to get the random number. The random number is stored in "cmddata".

Please refer to < Samples\API32VC\VC6\Sample07>.

```
int i, num, hic, errcode;
char cmddata[256];

num = DIC_Find();
for (i=0;i<num;i++)
{
	hic = DIC_Open(i, NULL);
	if (hic >= 0) break;		// Stop searching once one is found
}
if (i == num) return;

DIC_Set(cmddata, RANDOM_SIZE, BY_VALUE, 16, NULL);	// Set the random number length
errcode = DIC_Command(hic, RANDOM, cmddata);	// Get random number
for (i=0;i<16;i++) printf("%02X ", (BYTE)cmddata[i]);
printf("\n");

errcode = DIC_Close(hic);
```

10.9 Sample08: Super Password

The super password is for the developers and may never be shared with end users. Super password verification is required for specific dongle operations.

The "CHECK_SUPER_PASS" in the "DIC_Command" are used for verifying the super password. And the "GET_CARD_PRIVILEGE" macro will return the current privilege status of the dongle. Its return value is one byte in length. Every bit of the byte represents a different privilege status.

We defined the following status of privilege.

Table 10.9-1 Permission Levels

Macro	Permission Level
DICPR_NOPASS	Password not verified status
DICPR_SUPERPASS	Super password verified status
DICPR_REMOTEPASS	Remote update password verified status

The "SUPERPASS_STATUS" macro of "DIC_Get" shows if super password verification is successful. The return value of 1 represents successful verification, and 0 indicates super password verification failure. "REMOTEPASS_STATUS" macro detects whether the remote update password verification is successful or not. Please refer to < Samples\API32VC\VC6\Sample08>.

```
// Get privilege information of the dongle
    errcode = DIC_Command(hic, GET_CARD_PRIVILEGE, cmddata);
    if (DIC_Get(cmddata, SUPERPASS_STATUS, BY_VALUE, NULL))
        printf("super password is verified \n");
    else printf("super password is invalid \n");

    // Super password is 8 FF
    for (i=0;i<8;i++) cmddata[i] = (char)0xff;
    errcode = DIC_Command(hic, CHECK_SUPER_PASS, cmddata);

    // Get privilege information of the dongle
    errcode = DIC_Command(hic, GET_CARD_PRIVILEGE, cmddata);
    if (DIC_Get(cmddata, SUPERPASS_STATUS, BY_VALUE, NULL))
        printf("super password is verified \n");
    else printf("super password is invalid \n");

    // Update the super password
    // Put new super password (0 1 2 3 4 5 6 7) into the buffer
    // Parameter p3 of the DIC_Set has maximum (10 times) to try
    for (i=0;i<8;i++) buffer[i] = (char)i;
    DIC_Set(cmddata, SUPERPASS_DATA, BY_ARRAY, 10, buffer);
    errcode = DIC_Command(hic, SET_SUPER_PASS, cmddata);

    // Back to the normal user state
    errcode = DIC_Close(hic);
    errcode = DIC_Open(hic, NULL);

    // Try the new super password
    for (i=0;i<8;i++) cmddata[i] = (char)i;
    errcode = DIC_Command(hic, CHECK_SUPER_PASS, cmddata);

    // Get privilege information of the dongle
    errcode = DIC_Command(hic, GET_CARD_PRIVILEGE, cmddata);
    if (DIC_Get(cmddata, SUPERPASS_STATUS, BY_VALUE, NULL))
        printf("super password is verified \n");
    else printf("super password is invalid \n");
```

10.10 Sample09: Directory and File

This section covers directory and file operations. The complete example is in < Samples\API32VC\ VC6\Sample09>.

10.10.1 Formatting File System

The macro "FORMAT_CARD" in "DIC_Command" is to format the file system on the condition that the super password has been verified and the volume and software developer information has been set.

```
char data[256];

// 1. Change privilege to super user (Format operation is only available to super users)
for (i=0;i<8;i++) data[i] = (char)0xff; // By default, the super password is 8 FFs
errcode = DIC_Command(hic, CHECK_SUPER_PASS, data);

// 2. New volume and software developer information can and can only be set while formatting
DIC_Set(data, VOLUME, BY_ARRAY, 0, "DICSYSTEM");
DIC_Set(data, ATR, BY_ARRAY, 0, "DIC Co.");

// 3. Format
errcode = DIC_Command(hic, FORMAT_CARD, data);
```

10.10.2 Operations on Directory

"DIR_Set, DIR_Get" and "DIR_ID" of "DIC_CLASS" and "DIC_NAME" can set and read the directory ID, class, and name respectively, while "FILE_NO" defines the file number. "CREATE_DIR" and "LIST_DIR" in "DIC_Command" creates and lists the directory respectively. The list directory command enumerates all the directories until an error is returned indicating there are no more directories or files under the current directory.

```
// Create a directory with ID 0x1000, the long file name is "Dir1"
DIC_Set(data, FILL, 256, 0, NULL);  // Clear
DIC_Set(data, DIR_ID, BY_VALUE, 0x1000, NULL);
DIC_Set(data, DIR_CLASS, BY_VALUE, 0xff, NULL);
DIC_Set(data, DIR_NAME, BY_ARRAY, 0, "Dir1");
errcode = DIC_Command(hic, CREATE_DIR, data);

//    List directory, list all files and directories under the current directory
//    Note: list directory in the root directory. The first file is the root file, and //  the 2nd one is the software developer information file
```

```
for (i=0;i<255;i++)
{
      DIC_Set(data, FILE_NO, BY_VALUE, I, NULL);
      errcode = DIC_Command(hic, LIST_DIR, data);
      if (errcode != SCARD_S_SUCCESS) break;
      dirid = DIC_Get(data, DIR_ID, BY_VALUE, NULL);
      DIC_Get(data, DIR_NAME, BY_ARRAY, buffer);
      printf("%04x %s\n", dirid, buffer);
}
//   The macro SET_CURRENT_DIR of DIC_Command is used to enter a subdirectory; its parameter is the
//   ID of the subdirectory; SET_PARENT_DIR is used to go back to the upper-level directory without any
//   parameters

//   Enter directory 0x1100
DIC_Set(data, DIR_ID, BY_VALUE, 0x1100, NULL);
errcode = DIC_Command(hic, SET_CURRENT_DIR, data);
// Return to the upper directory, the value of the data is void
errcode = DIC_Command(hic, SET_PARENT_DIR, data);
```

10.10.3 Data File Operations

The macro "CREAT_FILE" of "DIC_Command" creates files, "WRITE_DATA" writes data into a file, "SET_CURRENT_FILE" selects a file, "READ_FILE" reads the selected file, and "REMOVE_FILE" deletes file(s). Note that executable files and internal files cannot be selected by SET_CURRENT_FILE without verification of super password.

The parameter settings for the operation of writing to files are special for the "DIC_Set" application, the convention is:

```
      DIC_Set(data, WRITE_DATA, write_size, write_offset, write_data);
```

The parameter settings for the operation of reading files for DIC_Set have the following format:

```
DIC_Set(data, READ_DATA, read_size, read_offset, NULL);
```

```
// Create file
DIC_Set(data, FILL, 256, 0, NULL); // Clear
DIC_Set(data, FILE_ID, BY_VALUE, 0x0033, NULL); // File ID
DIC_Set(data, FILE_CLASS, BY_VALUE, 0xff, NULL); // File class
DIC_Set(data, FILE_ATTRIBUTE, BY_VALUE, FILEATTR_NORMAL, NULL);      // Typically the property of the
// data file is set to FILEATTR_NORMAL
DIC_Set(data, FILE_SIZE, BY_VALUE, 20, NULL);                        // File size
DIC_Set(data, FILE_NAME, BY_ARRAY, 0, "File1");
errcode = DIC_Command(hic, CREAT_FILE, data);
```

```
// Write 20-byte information to the file
// Note: The new file just created is automatically set to be the current file,
// unnecessary to set it again.
DIC_Set(data, WRITE_DATA, 20, 0, "abcdefghijklmnopqrst");
errcode = DIC_Command(hic, WRITE_FILE, data);

//    Select the just created file with ID 0x0033
DIC_Set(data, FILE_ID, BY_VALUE, 0x0033, NULL);

errcode = DIC_Command(hic ,  SET_CURRENT_FILE ,  data);


//    Read the file and display its length and content

DIC_Set(data ,  READ_DATA ,  20 , BY_VALUE| 0 ,  NULL);

errcode = DIC_Command(hic ,  READ_FILE ,  data);

readsize = DIC_Get(data, READ_DATA, BY_VALUE, buffer);

// Delete the current file
errcode = DIC_Command(hic, REMOVE_FILE, data);
```

10.11 Sample10: Remote Update

See also "Chapter 5 Remote Update Management. You can find the complete example in < Samples\API32VC\VC6\Sample10>.

10.12 Sample11: Write and Execute Program

Users can develop their own programs inside the dongle. It is also one of the most important security features of the dongle. Feitian have provided IDE for users to develop and store their own programs inside the dongle.

We use the "Keil" development environment to develop internal programs of the dongle and then convert compiled "*.bin" files into C array by using "Tools\Binary > C" of the ROCKEY6 SMART IDE, and also insert it into the program, as well as burning the virtual card files produced by the Keil debugger simulator into the real device.

Executable files of the dongle must be written under the dongle root directory and they can only be accessed after super password verification. The complete example is in < Samples\API32VC \VC6\Sample11>.

10.13 Sample12: Double-Precision Point Calculation

This example will introduce how to perform float point calculations inside the card, and return the results to the external program. The complete example project locates in the < Samples \API32VC\VC6\Sample12>.

10.14 Sample13: Secure File Transfer

Regarding 'secure File Transfer", please also see "section 5.2 Secure File Transfer" in <<User Manual>> for details. The complete example project is in < Samples\API32VC\VC6\Sample13>.

10.15 Sample14: DES/3DES Encryption and Decryption

The ROCKEY6 SMART dongle with model "HID" has the features of DES and 3DES encryption and decryption. The steps for using them are as follows:

- Generating internal data files (super password is required)
- Writing DES or 3DES key (super password is required)
- The encryption and decryption procedures are all based on "file ID" and "key ID". That is, when encryption occurs, all data length that is not length in 8 multiples will be automatically filled into lengths in 8 multiples. Therefore, all encrypted data is length in 8 multiples. When decryption occurs, the dongle would restore the original data by removing all filled in data.

Notes:

The difference between DES key and 3DES key is their data length: the DES key length is 8 bytes.

In contrast, the 3DES key length is 16 bytes.

The data structure for writing in DES key and 3DES key is:

* The DES key length is 10 bytes – Key ID (1 byte) + key length (1byte, the value should be 8) + key (8 bytes)

* The 3DES key length is 18 bytes -- Key ID (1byte) + key length (1byte, the value should be 16) + key (16 bytes)

For a complete example, please refer to < Samples\API32VC\VC6\Sample14>. In this example, there is only a DES key and a 3DES key. In most cases, developers can freely set their file size and the keys accordingly.

10.16 Sample15: RSA Encryption/Decryption

In the "HID" model ROCKEY6 SMART dongle, the procedures for using RSA functions is as follows:

- ✓ Generating RSA key pairs (super password is required)
- ✓ Encrypting and decrypting data according to public key ID and private key ID.

Note: The key length is calculated in bits. Currently, it can support the key length of 1024 bits and 512 bits. The default value of "e" is 65537, and it is not modifiable. When data is encrypted, to prevent encrypting data bigger than "n", please set the most significant bit for plain text to zero.

The complete example is in < Samples\API32VC\VC6\Sample15>.

10.17 Sample16: Use Counter

Counter is a method that results in limiting the use times of the software by a developer. Once a counter is setup, the developer can control and define the use times decreasing progressively. The procedure is as follows:

- Setting up the counter (super password is required)
- The counter is decreased every time that the developer defined software procedure is performed.

If a developer sets the counter to zero, then it means the software can be used without times limitation. The default state for a formatted dongle is use without times limitation.

If the counter is deducted to "1", then the dongle cannot be used any more. It has to be sent back and reset by the developer. Usually, a developer would use the following processes to reset a dongle counter:

- ✓ After successfully verifying the super password, reset the counter to zero or any other numbers rather than "1".
- ✓ Formatting dongle, the counter would be automatically set to zero. However, this method is not normally used.

The complete example project is in < Samples\API32VC\VC6\Sample16>.

10.18 Sample17: Read Free Space

In order to manage the dongle memory space well, ROCKEY6 SMART provides the developer an interface to check the dongle remaining space. The complete example is in: < Samples\API32VC \VC6\Sample17>.

10.19 Sample18: Open Dongle with Management Code

The developer may have many different dongles from different manufactures therefore they need to manage their own dongle from the management code. ROCKEY6 SMART provides an API "DIC_FindByMgrCode". Developers may use this function to replace "DIC_Find" and also find the dongle with the same management code by using this function. This function increased a management code parameter compared to "DIC_Find" which is used by developers for inputting their own management code. Developers may fill the structure "DICST_ManagerCode" and send into "DIC_FindByMgrCode", as well as they can define a buffer as follows:

Unsigned Char mgnCode[ ]={0x56,0x00,0x02,0x00,0xee,0xff,0xee,0xff};//Make sure the order of the bytes, the management code here is 0056 0002 FFEE FFEE.

The complete example is in: < Samples\API32VC\VC6\Sample18>.

PART 3 Advanced Features

This part introduces RSA and DES encryption and decryption as well as APDU and COS system invoked usage. After over viewing this section the user will use ROCKEY6 SMART to improve program security level and also perform some flexible management.

Chapter11, System Call Function Usage

ROCKEY6 SMART provides a set of system call functions. The user can use them when they are compiling an external program, (executed in the smart card) such as file operation, fetching system information, and float point function etc.

Chapter12, Advanced Applications of File System

This chapter introduces the file filter and APDU command and its applications. Filter file is a special executable file that can filter some information. This chapter will describe how to create a filter file to reach maximum-security control.

Chapter13, COS System Call Reference

In the process of compiling the smart card program, the dongle default functions are used. Those functions are performed by system calls.

Chapter 11. System Call Function Usage

ROCKEY6 SMART provides five parts of the system call. (1) System function - such as input/output and COS security mechanism, etc., (2) File operation - such as open file, read file and write file. This part is fundamental to accessing the COS file system from external programming, (3) Float-precision calculation - such as some simple calculations; plus, minus, multiplication, division and some complicated extensive float-precision calculation, (4) Encryption and decryption - ROCKEY6 SMART provides three main encryption and decryption algorithms: RSA, DES, TDES and other algorithms can be extended by self-defined algorithms (5) Utility of system call - such as random number, request remote update marks. Some external applications can invoke the FEITIAN COS system service to extend their functions. For more detail, please refer to Chapter 13 COS System Call Reference.

Constitution of COS (Card Operating System) has been already introduced in this handbook. The following are problems that may appear in the development process:

Data file has an "internal use" attribute. It means this kind of file only can be operated in the dongle. System function provides six functions to perform the file operation. The file without marked "internal use" also can be accessed by an executable file. Please Note: The executable file can access the data file only when the file category of an executable file and a data file are in the same situation, and the security level of the executable file must be greater or equal to the data file.

If user performed some other manipulations after selecting data the selection will fail. They have to re-choose next time in such case.

11.1 Memory Management

When using KEIL to program ROCKEY6 SMART, because the ROCKEY6 SMART memory is pre-defined, users do not need to allocate the memory by themselves. Primarily, "data", "idata", and "xdata" are memories available to the user. "data", internal data area with direct addressing, is 128 bytes and has the fast speed to access variables; "idata", internal data area with indirect addressing, is 256 bytes and has ability to access all of the internal address spaces; "xdata" is the external data area with 64k bytes. In such manner, the big buffer should be defined in "xdata" area as following:

```
    unsigned char xdata buf[300];
```

As default, datum is stored in the data area therefore users can use the "xdata"area to share the datum. To transmit datum from one program to another you can use two methods, one is using "syscall", and the other is using the following method which is more convenient:

Program 1:

```
void main(void)
{
    unsigned char xdata *p=0;
    memset(p,1,128);
    set_response(128,p);
    exit();
}
```

Program 2:

```
void main(void)
{
    unsigned char xdata *p=0;
    set_response(128,p);
    exit();
}
```

The only difference between the two programs is code 2 has not performed a rewrite to buffer "*p*". Running code 2 generates 128 random numbers. (The numbers have not initialized in memory but are not from the random function) however code 1 generates 128 number "1". This is the basic principle of the sharing memory.

11.2 Fundamental Framework

"Input", "output" and "exit" in ROCKEY6 SMART program is defined as:

```
byte get_input(IN byte* pd, IN byte offset , IN byte mode, IN byte number)
byte set_response(IN byte len, OUT byte    *pd)
exit()
```

Note: In get_input function, mode represents type of input. For example, Replacing "get_input (&x1,0,1,1);" to "get_input(&x1,0,0,2);" all represent inputting 2 bytes however the results are different. The reason is that the order of bytes in the C51 program and the computer are opposite. If input "word (dword)" into the program therefore mode should be assigned to "1(2)".

C51 sample fac.c – a basic C51 program:

```
#include "sys_api.h"
// Factorial Function
int fun1(int a,int b,int *ab)
{
     int a1,b1;
     a1=a*a;
     b1=b*b;
     *ab=a1+b1;
}
// Main function
void main()
{
   int x1,x2,x12;
   // Get the first input variable
   get_input(&x1,0,1,1);
   // Get the second input variable
   get_input(&x2,2,1,1);
   // Results
   fun1(x1,x2,&x12);
   // Output
   set_response(2,&x12);
   //Quit
   exit();
}
```

The above example demonstrates the fundamental method to develop ROCKEY6 SMART by using Keil. Function "get_input"obtains data after calculation and uses function "set_response"to send data back. Eventually, invoke function exit to quit.

11.3 File Operations

There are five kinds of file operations in the ROCKEY6 SMART system call. They are shown in the following table:

Table 11.3-1 File Operation system function

System Function	Description
creat_file()	Create a file or directory
delete_file()	Delete the current file
open_file()	Open a file/directory or return to the up folder

write_file()	Write data into the file
read_file()	Read a file
get_file_infor()	Get a file information

The definitions are as follows:

```
byte creat_file(IN word wFileID, IN byte pbFileName, IN byte bAttrib, IN word wSize) ;

byte    delete_file()
byte    open_file(IN byte    *pd, IN byte    open_mode)
byte    write_file(IN word offset, IN byte *pd, IN word lenth)
byte read_file(IN word offset, OUT byte    *pd, IN word lenth)
byte get_file_infor(OUT byte *pd, IN byte mode)
```

Notes:

1) The "create_file" function is used for creating a file. When the file does exist you cannot create another. If creating a data file and it is a current file, users do not need to use the "open_file"function.

2) "Open_file" function is used for setting the current file to read, write and delete later.

3) "Write_file" function is used for writing file by offset, data and length. If the offset and the length are greater than the length of the file then the length will increase automatically, however the input of the offset should be less than the length of the file.

4) The "Read_file" function is similar to the "write_file" function, however if the offset and the length are greater than the file length, only the valid part of the input file can be read into the buffer. If users want to know the actual length of the input, they can use the "get_file_info" function to calculate the size (Actual input size equals File length minus offset) and it will also fail when the offset is greater than the file length.

C51 example "file.c", a complete file operation program:

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

void main(void)
{
    byte xdata is;
    word xdata fileID;
    byte xdata result[50];
    byte xdata info[110];
    word id;

    id=0x3f0a;
    // Input 102 bytes ,the first 2 bytes are the file ID, the rest are the content
```

```
    get_input(info,0, 0,102);
    // Get file ID
    memcpy(&fileID, info, 2);
    // Open a directory by using ID
    is = open_file(&id, 0);
    // If the directory does not exist
    if(is!=0)
    {
        // Create a new directory
        creat_file(id, "myfile",0x20,0xaa);
        //Enter the directory just created
        open_file(&id,0);
    }
    // Open the file
    is= open_file(&fileID, 0);
    // If the file does not exist
    if(is!=0)
    {
        // Create the file
        creat_file(fileID, "efFile", 0x00,0x64);
    }
    // Input the data into the dongle 0 is the file beginning; 100 is the file length
    write_file(0, info+2,100);
    // Read 20 bytes from the offset being at 80 byte
    read_file(80, result,20);
    // Delete this file
    delete_file();
    // Return to the upper-level directory
    open_file(NULL,2);
    // Return the results
    set_response(20, result);
    // Quit
    exit();
}
```

In the above example, Use the "open_file" function to open a folder. If the return is non-zero, it means you cannot open it, and no such sub-folder exists in the current directory. If this happens then use the "create_file" function to create the folder. After creating this successfully, use the "open_file" function to set the folder as current. By similar operation, use the "open_file" function to open a file, if this fails then use the "create_file" function to create a new file, after that is current.

11.4 Calling Executables

The ROCKEY6 SMART "sys_recall" function is used for invoking an executable file from another. The definition is as follows:

```
byte sys_recall(IN byte value, IN byte *pd, IN word lenth, IN byte *pdnext);
```

The first parameter is an additional procedure when exiting the program. The second parameter is the file ID which needs to run and is different to the external executable file. This is the executable file ID rather than the file name. The last two parameters respectively are the length of input for the running program and the file content. After having invoked and then entering the second program, the first program quits and transmits the privilege to the next.

The following code "syscall.c" is an example of one external program calling another one to show how to use the '"sys_recall()" function.

When compiling the file "test.c", rename the executable file as "F2F4" (or any other names, as long as the calling name is changed accordingly). The program will pass the data of 0-199 to the next program for use.

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

void main(void)
{
	byte xdata i, y[200];
	//The ID of the executable call by this program
	byte fileID[2];
	get_input(fileID, 0, 0, 2);

	for(i=0;i<200;i=i+1)
	{
		y[i]=i;
	}
	//Call another external program
	sys_recall(5, fileID, 200, y);	// Quit the current program and enter to "test.c"

	set_response(200, y);
	exit();
}
```

The program test.c is as follows:

```
void main(void)
{
	int i;
	byte xdata info[200];
	get_input(info, 0, 0, 200);    // Get the passed data 0-199 from the previous program
	for(i=0;i<200;i++)
	{
		info[i]+=1;
	}
	set_response(200,info);    // Output data is 1-200
	exit();
}
```

11.5 System Information and Security Mechanism

ROCKEY6 SMART provides the COS system functions and the relative functions of the security mechanism. These functions are rather simple to use. Their return information is more than their inputs. Therefore, it is necessary to set some variables for reading this information before calling them. There are no restrictions for the types of variables. The following is the definitions:

```
byte get_hard_infor(byte *pd);
byte get_sys_infor(byte *pd);
byte set_class(byte value);
byte get_class(byte * pd);
byte get_status(byte *pd);
```

- "get_hard_infor" is used for obtaining hardware information, such as the manufacture time and the serial number.
- "get_sys_infor" is used for obtaining system information, hardware version, and user code.
- "set_class" & "get_class" are used for setting and getting current security information.
- "get_status" is used for getting external password verification status.

"version.c" shows how to get hardware information and system information：

```
#include "sys_api.h"
void main(void)
{	byte xdata back[200];
```

```
// manufacture date and time (4bytes); serial number (4bytes)
get_sys_infor(back);       //len:8

// hardware version(4 bytes); COS version; Region code (2 bytes)
// Agent code (2 bytes); user code 1(2 bytes); user code 2(2 bytes)
    get_hard_infor(back+8);   //len:16

set_response(24, back);
exit();
}
```

"read_info.c" shows how to set and get current security information and external password verification status：

```
#include "sys_api.h"

void main(void)
{
    byte class;// file class, current external verification status
    byte security,security2;// system security level
    byte result;      // return value, 0 for successful operation
    byte xdata info[128];

    result=0;
    get_class(info);
    class=info[0];
    if(class!=0xFF)// If the program classification is not 0xFF
    {
      result=1;
      goto end;
    }

    security=0x06;
    set_class(security);
    get_class(info);
    security2=info[1];
    if(security!=security2)
    {
      result=2;
      goto end;
    }

    get_status(&class);
    if(class!=0)
    {
```

```
    result=3;
  }

end:
set_response(1, &result);
exit();

}
```

11.6 RSA Encryption/Decryption

The ROCKEY6 SMART supports internal RSA system calls. The difference between supporting internal RSA and external API invoked are (1) offering internal C51 directly invoked and (2) creating a correct encryption key without super password verification.

```
byte rsa_gen_key (word pubKey,word KeyLen,word PriKey)
byte rsa_enc(IN word fileID, IN word length, IN    OUT void *data)
byte rsa_dec(IN word fileID, IN word length, IN    OUT void *data)
```

- "rsa_gen_key" can be used to generate encryption keys. As default, the private key generated by "rsa_gen_key" cannot be read by any manipulation and can be created in any size internal file within the private key which is named as the same as the original one, that means this file will be replaced by the encryption key. It can be read by an internal program.
- "rsa_enc" is used for encrypting public key.
- "rsa_dec" is used for encrypting private key.

Actually, they are pair of reciprocal algorithms.

Program "rsa.c" demonstrates how to invoke RSA system call.

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

void main()
{
int rPubFileID=0x1002;    // Rsa public key file ID
int rPriFileID=0x1004;    // Rsa private key file ID
word rKeyLength=1024;    // Length of Rsa key space
unsigned char xdata tData[128]; // data to be encrypted
```

```
unsigned char xdata outData[256]; // output data
int ret;

    // Obtain data
    get_input(tData,0,0,128);

    // Generate key-pair
    ret=rsa_gen_key(rPubFileID,rKeyLength,rPriFileID);
    if(ret!=0)
        goto END;
    // RSA encryption
    ret=rsa_enc(rPubFileID,128,tData);
    if(ret!=0)
        goto END;
    memcpy(outData,tData,128);
    // RSA decryption
    ret=rsa_dec(rPriFileID,128,tData);
    if(ret!=0)
        goto END;
    memcpy(outData+128,tData,128);

    set_response(254,outData);
    exit();

END:
    set_response(2,&ret);
    exit();
}
```

11.7 DES Encryption/Decryption

The ROCKEY6 SMART C51 system provides DES and 3DES encryption/decryption algorithms which are:

```
byte des_enc(IN word fileID,IN byte keyID, IN word length, IN    OUT void *data) // DES encryption function
byte des_dec(IN word fileID,IN byte keyID ,IN word length, IN    OUT void *data) // DES decryption function
byte tdes_enc(IN word fileID,IN byte keyID, IN word length, IN    OUT void *data) //3DES encryption function
byte tdes_dec(IN word fileID,IN byte keyID, IN word length, IN    OUT void *data) //3 DES decryption function
```

- Parameter fileID is the key file ID; keyID is the key ID; length is encryption/decryption, length here is 8 integral multiples, data is the buffer to store the input and output data. The buffer will be covered after the function is performed.

- DES and 3DES algorithms are the same as in API. The following codes demonstrate how DES & 3DES

work.

The following programs des.c and tdes.c demonstrate how to encrypt and decrypt data using DES and 3DES algorithms.

DES encryption and decryption:

```
#include "sys_api.h"
#include <string.h>
void main()
{
	unsigned char key[10]={1,8,1,2,3,4,5,6,7,8}; // default key
	unsigned char xdata dData[129]; // data to be encrypted
	int i;
	int ret;
	byte Enc;
	creat_file(0x1122,0,0,10); // create key file
	write_file(0,key,10);// write key
	// obtain parameters
	get_input(&Enc,0,0,1);
	get_input(key+2,1,0,8);
	get_input(dData,9,0,128);

	if(Enc==0) // encryption and decryption
		dData[128]=des_enc(0x1122,01,128,dData); // use key 1122; keyID is 01
	else
		dData[128]=des_dec(0x1122,01,128,dData);

	set_response(129,dData);
	exit();
}
```

3DES encryption and decryption:

```
#include "sys_api.h"
#include <string.h>
void main()
{
	unsigned char key[18]={1,16,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}; // default key
	unsigned char xdata dData[128]; // data to be encrypted
	byte Enc;
	creat_file(0x1122,0,0,18); // create key file
	write_file(0,key,18);// write key
	// obtain parameters
	get_input(&Enc,0,0,1);
```

```
    get_input(key+2,1,0,16);
    get_input(dData,17,0,128);

    if(Enc==0) // encryption and decryption
        dData[128]=tdes_enc(0x1122,01,128,dData); // use key 1122; keyID is 01
    else
        dData[128]=tdes_dec(0x1122,01,128,dData);

    set_response(129,dData);
    exit();
}
```

11.8 Obtaining Random Number

Random number can be used in many programs. ROCKEY6 SMART provides a random generator which can generate random numbers, at most 16 bytes. The following example demonstrates the generator. If the size is greater than 16 bytes, the program only returns the first 16 bytes.

```
#include "sys_api.h"

void main(void)
{
    byte address[16];
    byte xdata len;
    get_input(&len,0,0,1);
    // generates random numbers according to the length
    get_rand(address,len);
    set_response(len,address);
    exit();
}
```

11.9 Using Float Function Libraries

There are basic float and complicated external float libraries in the ROCKEY6 SMART system call. They are all in double precision. The ROCKEY6 SMART uses the math coprocessor to complete the operation, the performance is better than the C51 internal float processor; therefore we recommend users only use double precision.

The definition of double precision should be as byte "a[8]" because Keil does not support double precision. Double and float type are the same, all 4 bytes.

Program "double.c" demonstrates how to use the basic float library.

```
#include "sys_api.h"
#include <string.h>
void main()
{
unsigned char a1[8]={0xA8,0x57,0xCA,0x32,0xC4,0x71,0x24,0x40};//10.2222
unsigned char b1[8]={0xF7,0x06,0x5F,0x98,0x4C,0x55,0x15,0x40};//5.3333
unsigned char result[72];
memset(result,0,72);
double_add(a1,b1,result);// plus
double_sub(a1,b1,result+8);// minus
double_mul(a1,b1,result+16); // multiply
double_div(a1,b1,result+24); // division
//Only change the 1st byte of the 3rd parameter. 8 bytes are used here for convenience
double_compare(a1,b1,result+32);// compare
double_abs(a1,result+40);// absolute value
double_sin(a1,result+48); // sine value
double_cos(a1,result+56); // cosine value
double_sqrt(a1,result+64); //extract
set_response(72,result);
exit();
}
```

Program "extdouble.c" demonstrates how to use the external float library.

```
#include "sys_api.h"
#include "string.h"
void main()
{
unsigned char a[8],b[8];
// define an array
unsigned char xdata uRet[105];
memset(uRet,0,sizeof(uRet));
// get parameters
get_input(a,0,0,8);
get_input(b,8,0,8);
// float calculaiton double_asin(a,uRet);        // arc-sine
double_acos(a,uRet+8);   // arc-cosine
double_atan(a,uRet+16);        // arc-tangent
double_sinh(a,uRet+24);        // double sine
double_cosh(a,uRet+32);        // double cosine
double_tanh(a,uRet+40);        // double tangent
double_ceil(a,uRet+48);   // Minimum integer number greater than double
```

```
double_floor(a,uRet+56); // Maximum integer number less than double
double_exp(a,uRet+64);   // double square
double_log(a,uRet+72);    // logarithm
double_log10(a,uRet+80); //common logarithm
double_mod(b,a,uRet+88); //modulus

//Return results
set_response(104,uRet);
exit();
}
```

The program convert.c demonstrates how to transform between the float point number and the integer.

```
#include "sys_api.h"
#include <string.h>
void main(void)
{
    unsigned char a1[8]={0xF2,0xB0,0x50,0x6B,0x9A,0x1F,0x30,0x40}; //16.12345
    long c1;
    unsigned char ret[12];
    double_2_int(a1,&c1);
    int_2_double(&c1,a1);
    memcpy(ret,&c1,4);
    memcpy(ret+4,a1,8);
    set_response(12,ret);
    exit();
}
```

11.10 Release Management

ROCKEY6 SMART provides software version management functionality in the C51 internal system call, thus logical control is executed within the dongle with high security. We can combine the internal program with these system calls.

ROCKEY6 SMART provides timer, counter and obtaining remote mark system call. They are defined as:

```
byte start_timer ()
byte get_timer (OUT dword * timer)
byte set_counter(byte counter)
byte step_counter()
byte get_counter(byte* counter)
byte get_remote_tag (OUT dword *Tag)
```

Program can trigger timer by start_timer. get_timer can be used for getting the interval (in 200MS).

The program timer.c demonstrates how to use the timer.

```
#include "sys_api.h"

void main()
{
// input data for the loop
unsigned int iTimer;
unsigned long i;
unsigned long dTimer; // return run time
get_input(&iTimer,0,1,1);

// start timer
start_timer();

for(i=0;i<iTimer;i++)
{

}

// get run time
get_timer(&dTimer);

// return
set_response(4,&dTimer);
exit();
}
```

In the program, users can use "set_counter" to set a number. Every time it is invoked it performs "step_counter".

The dongle will be locked when the count equals zero.

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

void main(void)
{
	word counter;
	int i;
	unsigned long count=0;
	// set number to 5
	set_counter(5);
	// get times of the loop
	get_input(&counter, 0, 1, 1);
	for(i=0;i<counter;i++)
	{
		// decrease counter
		step_counter();
	}
	// get counter
	get_counter(&count);
	// return counter
	set_response(4,&count);
	exit();
}
```

In the program, users can use "get_remote_tag" to get a remote update mark and then the internal code can control the program logic by the mark. Different marks match the corresponding codes.

Example "Remote.c" demonstrates how to get a remote update mark:

```
#include "sys_api.h"

void main()
{
dword dwTag;
// get remote update flag
get_remote_tag(&dwTag);
// make some control logics here
//...
// output remote update flag
set_response(4,&dwTag);
exit();
}
```

Chapter 12. Advanced Applications of File System

12.1 Filter File

Filter file is a special executable file. Its name uses the reserved values of 7816-4. Therefore, the difference between the filter file and the ordinary executable file can be recognized by their names.

Just as the name indicates, filter file refers to a file with filtering functionality. It can filter some specific information. In this case, the filtered information is information that the PC sends to the ROCKEY6 SMART. It is also called "APDU".

Filter file has the following special features:

- ✓ If super password verification occurs before the operation, then all filter files are worthless.
- ✓ The input buffer of the filter file can only start from 0. Namely, the first parameter of "input" instruction is "0".

12.2 APDU

APDU stands for "Applying Protocol for Data Unit". It is the fundamental data structure for communication between the PC and IC card. Because ROCKEY6 SMART is a security product that is based on the IC card, all layer communication between the PC and IC card are proceeded via data switching. The API provided to developers is the interface of the APDU instruction package. If software developers need to develop filter programs, then it is necessary to know when a read file command is issued and what kind of data the IC card will receive. Here we will describe the APDU command formats of the layer communication between ROCKEY6 SMART and the IC card.

12.2.1 APDU Structure

APDU can contain both command messages and response messages sent from the PC to the card or vice versa. Both command messages and response messages can carry data, except for the command message header.

12.2.2 APDU Command Message

APDU Command Messages are messages to the card from the computer. Their structure is:

CLA	INS	P1	P2	Lc	Data field	Le

Figure 12.2-1 APDU Command Message Structure

The meaning of each field is as follows:

Table 12.2-1 APDU Command Message Fields

Category	Field	Meaning
Command Message Header (Required)	CLA (1byte)	Instruction code
	INS (1byte)	Instruction categories
	P1 (1byte)	Instruction parameter 1 (interpreted by the specific instruction)
	P2 (1byte)	Instruction parameter 2 (interpreted by the specific instruction)
Others	Lc (1byte)	The length of data field (the length of data field after the "Lc" is variable)
	Data field	Data content
	Le (1byte)	Expected max length of data field in response message (It is also the expected return data length)

12.2.3 APDU Response Message

This is the message to the computer from the card. Its structure is:

Data field	SW1	SW2

Table 12.2-2 APDU Response Message Structure

SW1 and SW2 are the command message tails that must be present. If an error occurs in the IC card, the error code is returned through those two bytes.

Data field is actually the return data from the IC card to the PC. If the return data length is uncertain, then a response message is sent to the PC about the return data length. After that, the data message will be sent.

12.3 ROCKEY6 SMART APDU Command Set

The ROCKEY6 SMART APDU command message structures are introduced below. They are the foundations for writing filter programs.

12.3.1 Creating File

CLA = 0x00
INS = 0xE0
P1 = 0 Create volume label
= 1 Create file or directory
P2 = (create volume), it indicates ATR file length (max. 15); not used otherwise.
LC = the length of data field (assume these data is stored in a Buffer with BYTE type).

Descriptions of the data in data field:

1) When P1 = 0 (Create volume label)

Buffer[0] - Buffer[15]: Name of volume label
Buffer[16] - Buffer[30]: ATR data

Note:

The volume must have a name.

The volume can only be created by the software developer. The end user is not authorized to create it.

When creating a volume, the COS will remove the remote update password and update tag.

2) When P1 = 1 (create directory or file)

Buffer[0] - Buffer[1]: File ID number, one WORD, low byte precedence
Buffer[2]: File class
Buffer[3]: File properties
Buffer[4] - Buffer[5]: File length, one WORD, low byte precedence
Buffer[6] - Buffer[22]: Optional long file name

Note:

Long file name is optional, if it is not necessary, then clear Buffer[6] - Buffer[23].

Super password verification is required to create an executable file.

12.3.2 Selecting File

CLA = 0x00

INS = 0xA4
P1 = 0 (Search file by ID)
= 1 (Search file by name)
= 2 (Directly select the upper directory)
P2 = Not used
LC = length of data in data field (Assume this data is stored in a buffer with BYTE type)

Description of data in data field:

1) When P1 = 0 (Search files by their IDs)

Buffer[0] - Buffer[1]: File ID number, one WORD, low byte precedence

2) When P1 = 1 (Search file by name)

Buffer[0] - Buffer[15]: Filename

3) When P1 = 2 (Directly select the upper directory)

No data.

12.3.3 Reading Binary

CLA = 0x00
INS = 0xB0
P1 = High byte of offset
P2 = Low byte of offset
LC = buffer length (byte number of data to be read in)

No data field, but there should be a buffer to store the data to be read.

Note:

Every time the length of data to be read should not exceed 250 bytes.

12.3.4 Writing Binary

CLA = 0x00
INS = 0xD0
P1 = High byte of offset
P2 = Low byte of offset
LC = Number of bytes to be written to file

The data in data field is the content to be written into the file.

Note: The maximum data length that APDU can write is 128 bytes. For data of more than 128 bytes, the data has to be broken into 128-byte blocks.

12.3.5 Getting Random Number

CLA = 0x00
INS = 0x84
P1 = Not used
P2 = Not used
LC = expected byte number of random number (currently 6 bytes max)

No data field.

12.3.6 Reading Current Directory Record (Listing Directories)

CLA = 0x00
INS = 0xB2
P1 = record number (number of selected directory)
P2 = Not used
LC = Buffer length (must be 24 bytes)

There is a 22-byte buffer attached.

Buffer[0] - Buffer[5]: for 6-byte file description input .

Buffer[6] - Buffer[21]: for 16-byte file name input (Clear in case of no file name).

12.3.7 Deleting File

CLA = 0x00
INS = 0xEE
P1 = (delete current directory; 1, delete current file)
P2 = Not used
LC = 0

12.3.8 Getting Current Directory/File

CLA = 0x00
INS = 0x18
P1 = 0, get current directory; If 1, get current file; if 2, get the upper directory
P2 = Not used
LC = Returned data byte number from dongle, must be 22

Returned Data:

Buffer[0] - Buffer[5]: for 6-byte file description input.

Buffer[6] - Buffer[21]: for 16-byte file name input (Clear in case of no file name).

12.4 Filtering APDU

12.4.1 Naming Filter File

All the commands described in Section 12.3 can be used as filter file names. The 2-byte "CLA.INS" can be converted to a 4-byte file name, where the 4 high bits and the 4 low bits of every byte are converted into one byte respectively. Therefore, each byte of the file name to be discussed is a hexadecimal numeral. Another difference between the filter file name and the common executable files is that the filter file can use long file name. Its convention is:

CLA.INS. [C class ID/F file ID]

The dot in the middle is to distinguish each part of the file name and it has no affect. Items in the square brackets are optional, that is, the long file name may contain either its file class or ID.

Take executable file 00B0 as an example. When the system receives CLA.INS 00B0 of the APDU command, it calls executable file 00B0, which filters the read file operation.

00B0CFE implies that, when the system receives CLA.INS 00B0 and the data file to be accessed is of FE class, the executable file is called.

00B0F3002 indicates that, when the system receives CLA.INS 00B0 and the data file ID is 3002, the executable file is called.

12.4.2 Multiple Filter Files

Before discussing multiple filter file applications, it is necessary to review the return function of the executable programs. Normally, a program returns with exit. Its second parameter indicates whether there is returned data. If the parameter is "0x61", COS will return the data directly to the host computer when the program exits. If it is "0x90", no data is to be returned and the COS will not return any data to the host computer.

If there are several filter files, when receiving an APDU, COS would execute the APDU in batch in the following order: (XX is file category, yyyy is file ID)

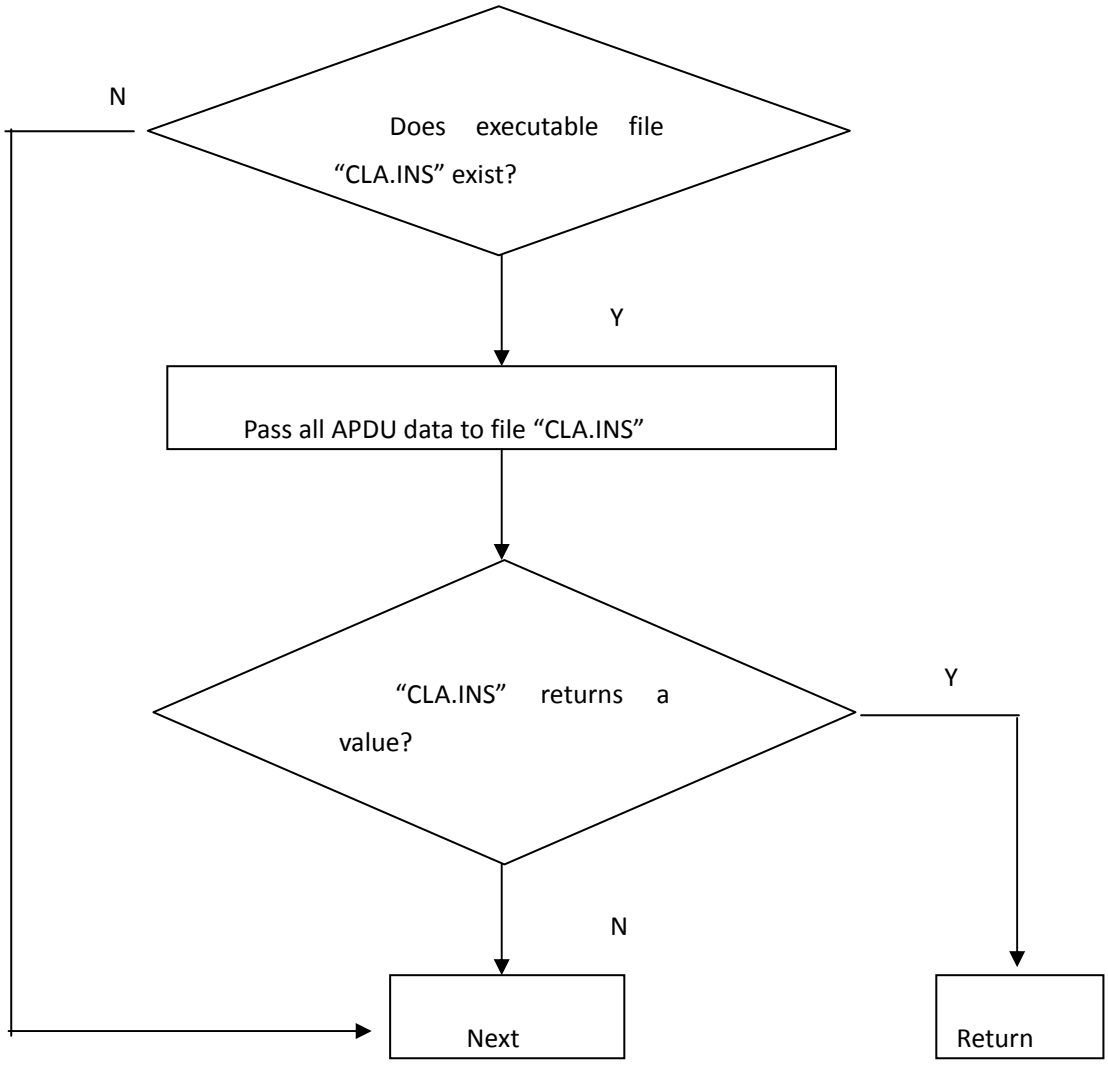


Figure 12-1 Process for Multiple Filters

(2) Executes the APDU in the same way as (1), only names of the executable files are different. The COS checks whether the executable file CLA.INS.CXX exists. If not, jump to the next step; otherwise, transfer the APDU to the file, and check its return function parameter. Just in case there is data to be returned to the host computer, execute the return operation, and if not, jump to the next step.

After the 3rd step, if the program does not exit, COS will execute the corresponding instruction according to

"CLA.INS".

The above are all the operations when COS receives an APDU. After the completion of every step, it will transfer APDU to the next step. And in case that any error occurs in any of the chained programs or there is data returned to the host computer, COS will terminate the program. When transferring data to filter files, the COS places the entire APDU in the filter file input buffer (corresponding to the input instructions of LC programs), instead of merely delivering the data to the filter file. This is different from dealing with other executable files.

12.4.3 Filter Read File Sample

In previous sections, we mentioned that the COS interprets all information (APDU) to ROCKEY6 SMART from the host computer and then executes the interpreted commands. For example, API sends to ROCKEY6 SMART a command for reading file in the following codes. Assume that the file with ID 0x0002 is a common data file.

```
DIC_Set(data, FILE_ID, BY_VALUE, 0x0002, NULL);
errcode = DIC_Command(0, SET_CURRENT_FILE, data);

//    read the file and display the file length and content
DIC_Set(data, READ_DATA, 10, BY_VALUE|0, NULL);
errcode = DIC_Command(0, READ_FILE, data);
readsize = DIC_Get(data, READED_DATA, BY_VALUE, buffer);
```

Line 4 and 5 set two the input data before read file is executed: 10 is the data length to be read and 0 is the shift value. Referring to Section 4.3.3, you will find the host computer sends 3 APDU to ROCKEY6 SMART: 0x00B0, 0x0000, 0x0A, in total 5 bytes.

When receiving the APDU the COS first checks if executable file 0x00B0 exists. If it exists, the APDU is transferred to executable file 0x00B0 as a string, and then the executable file runs. If the file 0x00B0 does not exist, the COS will read the 10 bytes of data from the file 0x0002 and return it to the host computer. If the executable file 0x00B0 does exist, it filters the command message of the APDU. It is up to the programmer to define how to process the command message in the program, such as to open a data file with ID 0x0020 and get 10 bytes of data starting from byte No.10 and then return. The read data can be encrypted before it is returned to the host computer. The host computer in return decrypts such data.

For example, in ROCKEY6 SMART, there is an executable file named 00B0F0002:

```
    word i;
    word fileID;
    byte xdata be[50];
```

```
        fileID=0x0002;
    open_file(&fileID, 0);
    read_file(0, be, 20);
    for(i=0; i<20; i++)
    {
        be[i]+=1;
    }
set_response(20, be);
exit();
```

This filter file filters the reading operation to the data file with ID 0033. The COS transfers the 5 bytes of APDU to the filter file. Then the data file with ID 0002 is read with the same shift and access byte number, and every byte accessed is added with 1, and then returned. Of course, it can make other operations to the command message. If the filter file is available in ROCKEY6 SMART, we can view the filtered output by double clicking the file with ID 0002 in IDE, which is different from the content in the original file. After super password verification, all the filter files will be disabled; the display is the original content of the data file.

12.4.4 Filter List Directory

The APDU to read the current directory is 00B2. If it is necessary to prevent the unauthorized person to view the contents of the dongle, you may filter the list directory to return an error code if the list directory is attempted. The following code will return the error code 0x6982:

```
void main()
{
   word sw = 0x6982;
   set_response(2,&sw);
   exit();
}
```

Save the file in dongle under the name 00B2. Without the super password, opening a dongle wil return an error message; it is a prompted internal error.

The name can be changed to 00B2FYYYY in order to protect a specific directory.

12.4.5 Filtering Selected Files

The APDU to select file is 00A4. It is necessary to select file(s) before to read/write/delete files. Therefore, filtering file selection can perform the overall protection of files. In addition, filtering directory file can protect all the files under the directory. If the conditions are not met, it is not possible to list directories.

12.4.6 Filter Write File

It is not necessary to require super password verification for writing non-executable files. If there are some important data files and certificates in the dongle and they are not internal files, you can have the files write protected. For example, to protect data files with Class 0x32 or ID 0x7022 against any unauthorized write and only you yourself can write properly. You may design in this way: filter the file write command, the first 8 bytes of the data from the APDU are treated as the file password. First check for the file password, if incorrect, file write is rejected; if correct, write the data after the 8-byte password to the file. In this case, the maximum length of data to be written is 120 bytes. Following is the example code:

```
void main()
{
    byte    xdata para[41],length = 41,i;
word fileID;
byte key[8];
get_input(para,0,0,41);

    //Default password
    key[0]=0x30;
    key[1]=0x31;
    key[2]=0x32;
    key[3]=0x33;
    key[4]=0x34;
    key[5]=0x35;
    key[6]=0x36;
    key[7]=0x37;
    //Verify if the password is correct
    for(i=0; i<8; i++){
        if(key[i]!=para[i]) {
            exit();//Incorrect password; quit the application
        }
    }
fileID= 0x0005;
i= open_file(&fileID,0);
if(i==0x05)
{
    creat_file(fileID, NULL, 0, 0x2500);
    open_file(&fileID, 0);
}
write_file( 0x0000, para+8, 33);
exit();
```

```
}
```

Naming the file as 00D0F0005 after compilation, this indicates to proceed filtering once writing to file with ID 0x0005. It will avoid any unauthorized writing to this sensitive file.

Chapter 13. COS System Call Reference

The following is all the system call functions descriptions:

13.1 File Operations

Operations can be performed on the files and directories on the EEPROM.

13.1.1 creat_file

Table 13-1 creat_file

Function	Create a file or directory	
Syntax	byte creat_file(IN word wFileID, IN byte pbFileName, IN byte bAttrib, IN word wSize)	
Parameters	wFileID	File/Directory ID
	pbFileName	File/directory name buffer area (file name is in 16 bytes maximum. If there is no file name, then the first letter of the buffer should be clear to 0 or set to NULL pointer), such as "aaa".
	bAttrib	File/Directory attributes and security levels
	wSize	File length (counted in bytes, if it is directory, parameter is 0; low byte is at the left part, and high byte is at right part.)
Return	Return code, please see also section 13.10.	

The file ID can be represented by a word type variable as the parameter; it is not necessary to be a number. Please note the file / directory class is the same with that of the executable file. If a file is created, it is selected by default upon completion, and it is ready for read, write, or delete operations. If a directory is created, use "OpenFile()" to open the directory before entering it.

Following is the detailed description for each parameter.

13.1.2 File/Directory ID

When creating file in the root directory, "0000" (system reserved), 2F01 (ATR file, including manufacturer information), 3F00 (volume) and 3FFF (current directory) cannot be used. When creating file or directory in any sub-directory, please do not use 3FFF, because this ID represents the current directory.

13.1.3 Properties and Security Levels of File/Directory

The high part of this type represents the file / directory attributes. Following is their description:

Property	Flag	Description
Normal	0x00	Not any special attributes
Executable	0x10	The file is executable, but it can only be created by the main program, no other programs in the dongle.
Dir	0x20	Create a directory
Up-to-Ignore	0x40	It only works for the executable files. If it can work, it means the security levels of the executable file must be higher than the system security levels. In other words, if the security levels of the executable file is less than or equal to the system security levels, then the executable file cannot be executed.
Internal	0x80	If it is a data file, then it represents that the data file can only be accessed by the executable files in the dongle. External operations can only delete it rather than read and write after successful super password verification. Without successful super password verification, it is impossible for external programs to access the data file. If it is executable file, then it means the executable file has some hidden attributes. When all files are listed, if there is no successful super password verification, all executable files with the hidden attributes cannot be shown. It is an important way to protect executable files.

The file security levels are represented via the low byte of the "abbr". It contains 16 distinguished authorization levels. The lowest level is 0, and the highest is 15.

COS has its own security levels.

An executable file can only operate a data file with a lower or same security level as itself (read, write, create). The current system security level can be cleared when exiting and set it to the lowest level 0.

13.1.4 delete_file

Table 13-2 delete_file

Function	Delete an opened file
Syntax	byte delete_file()
Parameters	None
Return	For the return codes, see section 13.10.

The internal operations are not provided with the interfaces for deleting directories. It can only be performed externally or used with the provided API.

13.1.5 open_file

Table 13-3 open_file

Function	Open file/directory or return to upper level	
Syntax	byte open_file(IN byte *pd, IN byte open_mode)	
Parameters	pd	The pointers for the file names or file IDs.
	open_mode	Open method 00 Sort in file/directoy ID, (pointer is pointing to file/directory ID, the lower byte is in the front). 01 Sort in file/directory name (at this time, the pointer is pointed to file/directory names) 02 directly open the upper directory. At this time, "pd" is NULL or other values; however, its value would be neglected.
Return	For the return codes, see section 13.10.	

If the directory names or IDs were being transferred, then they would enter the sub directory.

13.1.6 write_file

Table 13-4 write_file

Function	Write data into the file	
Syntax	byte write_file(IN word offset, IN byte *pd, IN word lenth)	
Parameters	offset	The shifting positions for the file data to be written in.
	pd	Pointer buffer, the initial address for the source data.
	lenth	The number of total bytes to be written in.
Return	For the return codes, see section 13.10.	

13.1.7 read_file

Table 13-5 read_file

Function	Read data from a file	
Syntax	byte read_file(IN word offset, OUT byte *pd, IN word lenth)	
Parameters	offset	The shifting positions of the file data to be read.
	pd	Pointer, points the address that stores the data.
	lenth	The number of the total bytes to be read.
Return	For the return codes, see section 13.10.	

Note: There is no "close file" operation to be displayed. Once the application program opens a new file, or the application program finished its execution, the previous opened file will be closed automatically.

13.2 Security Mechanism

13.2.1 set_class

Table 13-6 set_class

Function	Change the COS security state level	
Syntax	byte set_class(IN byte value)	
Parameters	value	Assigned security state level. (It could be 0x00-0x0f or 0xff. If the parameter is 0xff, the COS security level would change to the same security state level with the function "set_class".)
Return	For the return codes, see section 13.10.	

Note: Because the security level is related to the file class, when the COS security level changes, current system file class will change to the executable class accordingly. To execute it properly, the executable file level must be higher than or equal to the security level of "set_class" function.

13.2.2 get_class

Table 13-7get_class

Function	Get the current system security level
Syntax	byte get_class(OUT byte * pd)
Parameter	Pd address, points to the return value.
pd[0]	The current system file class
Pd[1]	The current system security level
Pd[2]	Application program security level
Return	For the return codes, see section 13.10.

13.2.3 get_Status

Table 13-8 get_status()

Function	Get system external verification state
Syntax	byte get_status(IN byte *pd)
Parameters	Pd address, pointing to the return value.
Return	For the return codes, see section 13.10.
	There are three return values for the current system external verification state: 0 Non-verified password state 4 Super password verification state 8 Remote update password verification state

13.3 System Services

13.3.1 exit

Table 13-9 exit()

Function	Stop execution and return to the main program.
Syntax	Exit()
Parameters	None
Return	None

13.3.2 sys_recall

Table 13-10 sys_recall

Function	Execution of self-terminated and call another application program.	
Syntax	byte sys_recall(IN byte value, IN byte *pd, IN word lenth, IN byte *pdnext);	
Parameters	value	Additional operations for stopping the program: * bit0 security level clear bit 0/1 – maintain/clear the application security level settings. * bit1 RAM clear bit 0/1 – maintain/clear the VM RAM (after the data passed back to the host machine). * bit2 used to choose the next file 0/1 – using file name/file ID to choose the next executable file. The range of its values is 0-7.
	pd	Address, points to the ID of the program to be called.
	lenth	The length of the buffer (it will be passed to the next application program)
	pdnext	The buffer starting address (it will be passed to the next application program)
Return	None	

13.3.3 set_response

Table 13-11 set_response

Function	Return the response data to the caller, and using "get_response" function to get the response data.	
Syntax	byte set_response(IN byte len, OUT byte *pd);	
Parameters	len	The length of the return data
	pd	Address, points to the storage place for return data
Return	None	

13.3.4 get_input

Table 13-12 ger_input

Function	Get the input parameter of the application program	
Syntax	byte get_input(IN byte* pd, IN byte offset , IN byte mode, IN byte number);	
Parameters	pd	Address, points to the data address
	offset	The shifting in every parameters
	mode	mode=0, BYTE type data, mode=1, WORD type data, mode=2, DWORD type data
	number	It represents the number of parameters with type "MODE".
Return	For the return codes, see section 13.10.	

Note: When MODE set to 1 or 2, the bytes are shifted inside COS. If we want to find a file with ID=0x0057, then we would input file ID to 0x0057. It will be shifted to the left-part right once using "get_input (&ID, 0, 1, 1). Inside the COS, we would then get the file with ID=0x0057. It will work compatibly with the previous versions.

13.4 System Information

13.4.1 get_hard_infor

Table 13-13 get_hard_infor

Function	Get system hardware information
Syntax	byte get_hard_infor(OUT byte *pd)
Parameters	Pd address, pointing to the return value.
pd[0..3]	Manufacture date and time
pd[4..7]	Hardware serial number
pd[7..11]	Shipping date and time
pd[11..15]	COS version
Return	For the return codes, see section 13.10.

13.4.2 get_sys_infor

Table 13-14 get_sys_infor

Function	Get system management information
Syntax	byte get_sys_infor(OUT byte *pd)
Parameters	Pd address, pointing to the return value.
pd[0..3]	Region code
pd[4..7]	Agent code

pd[7..11]	User code 1
pd[11..15]	User code 2
Return	For the return codes, see section 13.10.

13.4.3 get_file_infor

Table 13-15 get_file_infor

Function	Get relevant information of current directory or file	
Syntax	byte get_file_infor(OUT byte *pd, IN byte mode)	
Parameters	pd	Address, points to the return value
	pd[0]	FIle ID or directory ID
	pd[2]	File class or directory class
	pd[3]	File attributes and file security level or directory attributes and directory security level.
	Pd[4..5]	File size or directory size
	Pd[6..21]	File name or directory name
	mode	There are three values for choosing the directories or files: 0 get current directory 1 get current file 2 get upper directory
Return	For the return codes, see section 13.10.	

Here, the directory refers to the directory for holding the current executing program; the file refers to the files that are operated by the executing program. If the executing program does not operate any files, then the return information with parameter equals one and would be meaningless.

13.5 Double Precision Float Calculation

13.5.1 Addition, Subtraction, Multiplication, and Division

Table 13-16 Addition, Subtraction, Multiplication, and Division

Function	Calculate adding, subtracting, multiplying and dividing for double precision float numbers.	

Syntax	byte double_add(IN byte *a, IN byte*b, OUT byte *result);
	byte double_sub(IN byte*a, IN byte*b, OUT byte*result);
	byte double_mul(IN byte *a, IN byte*b, OUT byte *result);
	byte double_div(IN byte*a, IN byte*b,OUT byte*result);	

Parameters	a	Address, points to the first floating point number. It is the number to be added / subtracted / multiplied / divided.

	b	Address, points to the second floating point number.
	result	Address, points to a 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.5.2 sqrt

Table 13-17 sqrt

Function	The floating point number square root calculation	
Syntax	byte double_sqrt(IN byte *a, OUT byte *result);	
Parameters	a	Address, points to a floating point number
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.5.3 Sine and Cosine

Table 13-18 Sine and Cosine

Function	Calculating the value of sine and cosine for float numbers.	
Syntax	byte double_sin(IN byte *a, OUT byte *result) byte double_cos(IN byte *a, OUT byte *result)	
Parameters	a	Address, points to a floating point number, the number is a radian value.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.5.4 double2int

Table 13-19 double2int

Function	Double precision float numbers converts to signed 32-byte integer, and will take the integer part directly.	
Syntax	byte double2int(IN byte *a, OUT byte *result);	
Parameters	a	Address, points to a floating point number.

	result	Address, points to a 4-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.5.5 int2double

Table 13-20 int2double

Function	Signed 32-byte integer converts to double precision floating point number.	
Syntax	byte int2double(IN byte *a, OUT byte *result);	
Parameters	a	Address, points to a signed 32-byte integer.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.5.6 abs

Table 13-21 abs

Function	Absolute value for float point numbers	
Syntax	byte double_abs(IN byte *a, OUT byte *result);	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.5.7 compare

Table 13-22 compare

Function	Compare two float point numbers.	
Syntax	byte double_compare(IN byte *a, IN byte *b, OUT byte *result);	
Parameters	a	Address, points to the first float point number.
	b	Address, points to the second float point number.
	result	Address, points to a 1-byte buffer that is used for storing computation results.

Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.

There are three computation results: 0 (a ==b), 1 (a > b) and –1 (0xff) (a < b).

13.6 Float Library Extension

13.6.1 asin

Table 13-23 asin

Function	Calculating the value of asin	
Syntax	byte double_asin(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.2 acos

Table 13-24acos

Function	Calculating the value of acos	
Syntax	byte double_acos(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.3 atan

Table 13-25atan

Function	Calculating the value of atan	
Syntax	byte double_atan(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.

Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.

13.6.4 sinh

Table 13-26 sinh

Function	Calculating the value of sinh.	
Syntax	byte double_sinh(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.5 cosh

Table 13-27 cosh

Function	Calculating the value of cosh.	
Syntax	byte double_cosh(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.。
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.6 tanh

Table 13-28 tanh

Function	Calculating the value of tanh	
Syntax	byte double_tanh(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.7 ceil

Table 13-29 ceil

Function	Calculating the minimum integer greater than double.	
Syntax	byte double_ceil(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.8 floor

Table 13-30 floor

Function	Calculating the maximum integer less than the double.	
Syntax	byte double_floor(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.9 exp

Table 13-31 exp

Function	Calculating exp. function	
Syntax	byte double_exp(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.10 log

Table 13-32 log

Function	Calculating log function	
Syntax	byte double_log(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.11 log10

Table 13-33 log10

Function	Calculating the value of log10	
Syntax	byte double_log10(IN byte* a, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.6.12 fmod

Table 13-34 fmod

Function	Calculating the mod of two double precision floating-point numbers.	
Syntax	byte double_fmod(IN byte* a,double b, OUT byte* result)	
Parameters	a	Address, points to a floating point number.
	b	Address, points to two float numbers.
	result	Address, points to an 8-byte buffer that is used for storing the computation results.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.7 Encryption and Decryption Functions

13.7.1 des_enc

Table 13-35 des_enc

Function	des ECB encryption	
Syntax	byte des_enc(IN word fileID,IN byte keyID, IN word length, IN OUT void *data)	
Parameters	fileID	The ID of the file that is used for storing the key
	keyID	Storing the key ID; it is prohibited to store more than 256 keys in a data file.
	length	The length of input data
	data	Data address, used to store input data and return data.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.7.2 des_dec

Table 13-36 des_dec

Function	des ECB decryption	
Syntax	byte des_dec(IN word fileID,IN byte keyID ,IN word length, IN OUT void *data)	
Parameters	fileID	The ID of the data file that is used for storing keys.
	keyID	Storing the key ID; it is prohibited to store more than 256 keys in a data file.
	length	The length of input data
	data	data address, used to store input data and return data.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.7.3 tdes_enc

Table 13-37 tdes_enc

Function	3des ECB encryption	
Syntax	byte tdes_enc(IN word fileID,IN byte keyID, IN word length, IN OUT void *data)	
Parameters	fileID	The ID of the data file that is used for storing keys.
	keyID	Storing the key ID; it is prohibited to store more than 256 keys in a data file.
	length	The length of input data
	data	data address, used to store input data and return data.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.7.4 tdes_dec

Table 13-38 tdes_dec

Function	3des ECB decryption	
Syntax	byte tdes_dec(IN word fileID,IN byte keyID, IN word length, IN OUT void *data)	
Parameters	fileID	The ID of the data file that is used for storing the keys
	keyID	Storing the key ID; it is prohibited to store more than 256 keys in a data file.
	length	The length of input data
	data	data address, used to store input data and return data.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.7.5 rsa_gen_key

Table 13-39 rsa_gen_key

Function	Generating key pairs	
Syntax	byte rsa_gen_key (word pubKey,word KeyLen,word PriKey)	
Parameters	pubKey	The public key file ID
	KeyLen	The length of key space
	PriKey	The private key file ID
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.7.6 rsa_enc

Table 13-40 rsa_enc

Function	RSA encryption, the value of e is 65537.	
Syntax	byte rsa_enc(IN word fileID, IN word length, IN OUT void *data)	
Parameters	fileID	The ID of RSA public key.
	length	The length of input data
	data	data address, used to store input data and return data.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.7.7 rsa_dec

Table 13-41 rsa_dec

Function	RSA decryption, the value of e is 65537.

Syntax	byte rsa_dec(IN word fileID, IN word length, IN OUT void *data)	
Parameters	fileID	The ID of RSA private key.
	length	The length of input data
	data	data address, used to store input data and return data.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.8 Timer and Counter

13.8.1 get_timer

Table 13-42 get_timer

Function	Get the execution time once the dongle is activated.
Syntax	byte get_timer (OUT dword * timer)
Parameters	Return the address of the data; its unit is microsecond.
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.

13.8.2 start_timer

Table 13-43 start_timer

Function	Start timer
Syntax	byte start_timer ()
Parameters	None
Return	For the return codes, see section 13.10.

13.8.3 step_counter

Table 13-44 step_counter

Function	Counter decreases by 1.
Syntax	byte step_counter()
Parameters	None
Return	For the return codes, see section 13.10.

13.8.4 get_counter

Table 13-45 get_counter

Function	Get the value of the counter

Syntax	byte get_counter(byte* counter)	
Parameters	counter	address of the counter
Return	For the return codes, see section 13.10.	

13.8.5 set_counter

Table 13-46 set_counter

Function	set the times of the counter.
Syntax	byte set_counter(byte counter)
Parameters	None
Return	For the return codes, see section 13.10.

13.9 Others

13.9.1 get_rand

Table 13-47 get_rand

Function	Obtain a random number	
Syntax	byte get_rand(byte * pd,byte len)	
Parameters	Pd	Buffer pointer, is used for storing returned random number.
	len	Assigned length of random number (counted in byte)
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

The parameter can be a variable (either in "byte" type or in "word" type). Or it can simply use digits.

Note: It can assign any values to memory [0], but only 16 bytes would be return by the system. Thus, if more than a16-byte value is given, then the system can only return a random number in 16-bytes. There is no error message prompting.

13.9.2 get_remote_tag

Table 13-48 get_remote_tag

Function	Getting remote update tag	
Syntax	byte get_remote_tag (OUT dword *Tag)	
Parameters	Tag	Remote update tag
Return	For the return codes, see section 13.10. Once it is finished, the calculation result would be stored into the specified memory area.	

13.10 System Function Call Error Code

System call error codes

Table 13-49 System Function Call Error Code

Return Value	Description
0	Application program is correctly terminated.
1	Invalid file
2	The file already exists.
3	Insufficient memory space
4	Security requirement is not sufficient
5	File is not found
6	Memory operation error
7	Data error
8	Invalid parameter
9	Virtual machine overflow

Appendix A Glossary

A1 Main Program

The main program is the part of the software running on the host computer. It has complete control over software operation.

A2 Protected Program

The protected program refers to the part of the program that only runs inside the dongle. A program could run many of its sub programs inside the dongle. But only one protected program is activated each time. A protected program is a function independent program. That means the protected program has its own inputs and outputs corresponding to the encryption points in the main program.

A3 Protected Execution

Protected execution always follows the protected compile. It would fetch the program in the virtual card or real card.

A4 Executable File

Executable file refers to the binary code existing in the dongle after external program compiling. It can be executed inside the card.

Appendix B C51 Program Errors and SDK

B1 Errors in C51 Program

1. Please make sure the correct parameter type settings are used during the system call because KEIL never checks the parameter types. For example:

```
void main()
{
short fileid ;
open_file(fileid,0);//Correct
open_file(&fileid,0);//Incorrect, keil does not report error in compiling
}
```

2. The created file should be compatible with its previous version. Reversed creating file ID and the file size (low byte is located at left and high byte is at right). Following is an example:

```
void main()
{
   short fileID = 0x3f0a
         char info[100];
   get_input(info,100);
is= open_file(&fileID, 0);
   if(is!=0)
   {
              creat_file(fileID, "efFile", 0x00,0x0064);
    }
          write_file(0, info,100);    // Write file.
          read_file(80, result,20); // Read file, excursion is 80 and read 20 bytes.
}
```

3. When using "get_input()" function and the data mod is 1 or 2 (choosing the data type in short or int), the byte in the COS are reversed. Especially in selecting file ID case, the sample13 program inputs file ID=0x5700. This is actually the program accessing the ID=0x0057 file.
4. Another problem is about the "DIC_Get()" function. In sample11, when the COS result is 0x0019, we get "c = DIC_Get (buffer, 1, BY_VALUE |1, NULL); //get "c" from the running result". Please pay attention to "DIC_Get" parameters. In most cases, the error is caused from the incorrect byte sequence.
5. Due to some bugs in the Keil compiler, when you try to compile "dword dNum=255*255" the compiler will generate 0xfffffe01.
6. String initialization problem. Cannot directly set value to a pointer, such as: "char *p='str"; "should be replaced as "char p[]='str" ".

B2 ROCKEY6 SMART Developer's Kit

The user will find the ROCKEY6 SMART Developer's Kit in the ROCKEY6 SMART installation disc. The following table contains the general information for the ROCKEY6 SMART Developer's Kit ("setup.exe" is omitted):

Directory	Description
API32	32-bit API
API64	64-bit API
Docs	Manual etc.
Driver for Win98	USB MASS STORAGE common driver for Windows 98
Include	Included files
Net	ROCKEY6 SMART Net dongle DK
Samples	ROCKEY6 SMART samples
Support	3rd party tools, including Keil uVersion 2
Tools	IDE and user tools