

ROCKEY4 SMART User's Guide

Version 1.4

Copyright © 2009 Feitian Technologies Co., Ltd.
<http://www.FTsafe.com>

Feitian Technologies Co., Ltd. (“Feitian” for short) will do their best to keep the content of this document as accurate as possible. But Feitian will not take the responsibilities for any direct or indirect loss that may be caused by this document. The content of this document will be amended along with the updating of the product without notification.

Revision History:

Date	Version	Description
August 2009	1.4.0	1st Edition

Feitian Technologies Co., Ltd.

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, deducting 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 manual. 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.

CE Attestation of Conformity



The equipment complies with the principal protection requirement of the EMC Directive (Directive 89/336/EEC relating to electromagnetic compatibility) based on a voluntary test.

This attestation applies only to the particular sample of the product and its technical documentation provided for testing and certification. The detailed test results and all standards used as well as the operation mode are listed in

Test report No.: 70407310011

Test standards: EN 55022/1998 EN 55024/1998

After preparation of the necessary technical documentation as well as the conformity declaration the CE marking as shown below can be affixed on the equipment as stipulated in Article 10.1 of the Directive. Other relevant Directives have to be observed.

FCC certificate of approval



This Device is in conformance with Part 15 of the FCC Rules and Regulations for Information Technology Equipment.

Quick Start

- The ROCKEY4 SMART Evaluation Kit is available for developers. It includes a dongle, a package, a manual, a CD, and an extension cable etc. The evaluation dongle is completely the same as the formal version of dongle, except that its passwords are public (P1: C44C; P2: C8F8; P3: 0799; P4: C43B). After receiving the formal version, you can modify its passwords with the initialization tool, so that others cannot view and/or change the the content of your dongle.
- Install the ROCKEY4 SMART Developer's Kit. Run Setup.exe under the root directory of the CD. The Setup Wizard will guide you through the installation process. (For details, see Chapter 3.)
- Under directory Tools on the CD, you can find the Dongle Editor (Ry4S_Editor.exe) provided with the ROCKEY4 SMART dongle, which is a utility for you to make some necessary operations on the dongle (for details see Chapter 5). The envelope encryption tool, Tools\Envelop\RyEnvX.exe, enables you to encrypt the PE-structure files without programming work (for more information, see Chapter 2 in *ROCKEY4 SMART License Management*).
- With the function calling (API) protection method, you can embed the ROCKEY4 SMART API in the protected application program, making the most out of the dongle. In this way, the high-level security is achieved. Some example codes are provided to you to help understand the API encryption method. For details on the API encryption, see Chapter 6 and the sample programs under directory Samples on the CD.
- For some frequently asked questions and answers, see Chapter 8.
- For more information or latest updates, please visit: <http://www.ftsafe.com>

Contents

CHAPTER 1 INTRODUCTION.....	1
1.1 ABOUT ROCKEY4 SMART	1
1.2 SOFTWARE PROTECTION PRINCIPLES	2
1.3 ROCKEY4 SMART ADVANTAGES	2
1.4 CHOOSING AN APPROPRIATE ENCRYPTION SOLUTION.....	3
CHAPTER 2 HARDWARE FEATURES	5
2.1 INTERNAL STRUCTURE	5
2.2 HARDWARE INTERFACE.....	5
2.3 CONSIDERATIONS	5
CHAPTER 3 DEVELOPMENT KIT	6
3.1 CONTENTS OF THE CD.....	6
3.1.1 Tools.....	6
3.1.2 Development Interfaces	6
3.2 INSTALLING DEVELOPMENT KIT	7
3.3 UNINSTALLING DEVELOPMENT KIT	11
3.4 DRIVER INSTALLATION UNDER WINDOWS 98 SE	11
CHAPTER 4 BASIC CONCEPTS.....	13
4.1 PASSWORDS.....	13
4.2 HARDWARE ID	13
4.3 USER MEMORY AREA	14
4.4 MODULE CHARACTERS	14
4.5 MODULE PROPERTY CHARACTERS	15

4.6 ALGORITHM AREA	15
4.7 USER ID	15
4.8 RANDOM NUMBER	16
4.9 SEED CODE AND RETURN CODES	16
4.10 TIMER AND COUNTER	16
4.11 DONGLE CONFIGURATION UPDATE	16
CHAPTER 5 DONGLE EDITOR	18
5.1 INTRODUCTION	18
5.2 DESCRIPTION OF OPERATIONS	21
5.2.1 Entering Passwords	21
5.2.2 Editting	22
5.2.3 Testing	24
5.2.4 Timing and Number of Uses	25
5.2.5 Self-testing	27
5.3 SAVE YOUR WORK	27
CHAPTER 6 CALLING API FUNCTIONS	29
6.1 FUNCTION PROTOTYPE AND DEFINITION	29
6.2 ROCKEY4 SMART API SERVICES	32
6.3 ERROR CODES	48
6.4 BASIC APPLICATION EXAMPLES	49
6.4.1 Unencrypted Program – Step 0	50
6.4.2 Finding Dongle – Step 1	50
6.4.3 Opening Dongle – Step 2	51
6.4.4 User Memory – Step 3/Step 4	52
6.4.5 Generating a True Random Number with Dongle– Step 5	55
6.4.6 Seed Code – Step 6/Step 7	57

6.4.7 User ID – Step 8/Step 9.....	60
6.4.8 Setting Module Characters – Step 10/Step 11/Step 12	63
6.4.9 Dongle Cascading – Step 13.....	67
6.5 ADVANCED APPLICATION EXAMPLES	71
6.5.1 User Memory Area Applications.....	71
6.5.2 Seed Applications.....	81
6.5.3 User ID Applications	86
6.5.4 Module Applications	89
6.5.5 Dongles with Same UID for Different Software Products	95
CHAPTER 7 ROCKEY4 SMART HARDWARE ALGORITHMS.....	103
7.1 INTRODUCTION TO SELF-DEFINED ALGORITHMS.....	103
7.1.1 Instruction Format.....	103
7.1.2 Internal Algorithms & Application Interface.....	104
7.1.3 Difference Between Three Algorithms	105
7.1.4 API Interfaces of User Program	106
7.2 WRITING SELF-DEFINED ALGORITHMS.....	108
7.2.1 Writing Algorithm	108
7.2.2 Instruction Conventions.....	109
7.3 EXAMPLES OF USE OF USER-DEFINED ALGORITHMS.....	110
7.3.1 Basic Appliation Examples	110
7.3.2 Integrated Algorithm Application Examples.....	121
7.3.3 Advanced Algorithm Application Examples.....	137
7.4 CONSIDERATIONS	142
7.5 APPLICATION TIPS OF ENCRYPTION SOLUTION.....	142
CHAPTER 8 FAQS	144
8.1 COMMON WAYS OF DEALING WITH PROBLEMS.....	144

8.2 FAQs	144
APPENDIX A DIRECTORY STRUCTURE OF CD.....	147

Chapter 1 Introduction

1.1 About ROCKEY4 SMART

ROCKEY4 SMART is a software protection product. The device can be connected to the USB port of your computer. The software can be protected using the ROCKEY4 SMART encryption key against unauthorized duplicate, access or use. After encrypting and protecting your software with the dongle, when the protected software starts, an error message will be rendered and the protected software will stop running if the dongle device is not present, access to a module is restricted, or the predefined license expires,. At the same time, the ROCKEY4 SMART dongle also supports a comprehensive set of other protection mechanisms.

Unlike some other peer products, the ROCKEY4 SMART dongle is actually a mini-computer with a CPU, a memory and a tailored middleware, which interacts with applications. You can code some complicated algorithms in the dongle, and then call them in your programs to realize the encryption. This encryption method is recommended because it has high level of security and is hard to be cracked. In addition, the dongle is easy to use because the functions calling of the dongle is very simple.

The ROCKEY4 SMART dongle is provided with an envelope encryption tool. The envelope encryption tool (RyEnvX.exe) can be used to encrypt Windows PE files (i.e., .dll, .exe, and .arx files). The tool is very easy to use. It only takes you several seconds to encrypt such a file. If you do not have the source code, or you are not familiar with the APIs, it is a good idea to use the envelope encryption. If possible, the encryption performance can be greatly enhanced by calling the API functions in combination with the envelope encryption.

In this manual, we will describe the components of the encryption software one by one. The

following two parts are the main contents of this manual.

- 1) Encryption key editor (Ry4S_Editor.exe): It is a GUI tool for performing operations on the dongle. With the editor, you can not only read from or write to the dongle, but also carry out an algorithm operation or test the dongle. (For details, see Chapter 5.)
- 2) Encryption key APIs: They can be used to create strong protection solutions in a flexible way. In this manual, we use VC++ programs as examples to give extensive introduction. Besides, under the Samples directory in the CD, the sample programs in other languages are also provided. (For details, see Chapter 6.)

1.2 Software Protection Principles

The goal of software protection is reached by designing programs to require access to the dongle when they are running, so that the programs are dependent on the dongle and they cannot be duplicated due to the irreducibility of the dedicated chip of the dongle.

1.3 ROCKEY4 SMART Advantages

1 Small and Exquisite

The ROCKEY4 SMART dongle is small, exquisite, good-looking, and easy to carry.

2 High Speed

The running speed of the software encrypted by the ROCKEY4 SMART dongle is almost not affected. Even if the user defined an extremely complicated encryption algorithm in the dongle, the ROCKEY4 SMART dongle can still handle it in a very short period, and ensure the smooth running of the user's program.

3 Easy to Use

Both the API function calling encryption and the envelope encryption are designed in consideration of user convenience by simplifying user interfaces. In a short time period, the users can master the usage method of the ROCKEY4 SMART key, to cut down the time spent on software encryption.

4 High Encryption Strength

The ROCKEY4 SMART dongle is a brand-new strong strength encryption product. It employs a two-level password protection mechanism. With only the first level password, you cannot modify the special storage area in the dongle. Thus, the software developer and the end user can be granted with different rights. A time gate is built in the dongle to prevent software tracing. In addition, the software suppliers are allowed to define hardware encryption algorithms by themselves, thus raise the security of the encryption key to a new level.

5 High Reliability

The ROCKEY4 SMART dongle has a complet user management system. Different users will never obtain identical passwords. The hardware ID for each dongle is unique. The password and the hardware ID are burnt into the CPU, even the manufacturer cannot change them.

6 Comprehensive System Support

The ROCKEY4 SMART dongle supports various kinds of operating systems. The encrypted application programs support the following platforms: Windows 98 SE/Me/2000/XP/Server 2003/Vista/2008/Windows 7, Linux, and Mac OS.

7 Various Software Interfaces

Software interfaces are available for almost all popular development tools, such as PB, DELPHI, VB, VC, C++ BUILDER, C#, and Java, etc.

1.4 Choosing an Appropriate Encryption Solution

The strength of software protection depends not only on the dongle itself, but on how the developer uses it largely. You should make full use of the dongle to gain the greatest strength. The ROCKEY4 SMART dongle provides two encryption modes: the envelope encryption and the API calling encryption.

The envelope encryption can be achieved by the tool (RyEnvX.exe) under the directory “Tools\Envelop” on the CD. Just as its name implies, the envelope encryption is to add an envelope to some user files, which calls the ROCKET4 SMART encryption functions to realize the protection over software. . When running the program encrypted with an envelope, the envelope program section will automatically access the ROCKEY4 SMART dongle and determine if the program is allowed to

proceed or not, according to the result of the access. The envelope encryption directly encrypts and handles the compiled files. The advantages are that the developers need neither to learn a lot of encryption knowledge, nor to modify the source code. If you have no time to learn cryptography or you have no source code at hand, this is a good choice and convenient. However, the encryption strength level is not high. That is because the encryption handling is automatically completed by the programs, so it follows some rules and may be discovered; Moreover, it cannot encrypt the script language files (e.g., VBA) which cannot be compiled. These are the inherent defects of the envelope encryption.

The API calling encryption is to choose an appropriate language interface provided with the dongle to handle access to the dongle according to the language used for development by the developer. This mode allows you to make full use of the ROCKEY4 SMART dongle. In addition, the determination of encryption points and method is left to the developer. The security is lifted to a higher level, especially when the developer uses the internal algorithm functions of the dongle. But the API calling method should combine with the developer's source code, and require the developer to learn and master the API calling method of the ROCKEY4 SMART. It requests a higher starting level of the developer.

Chapter 2 Hardware Features

2.1 Internal Structure

The kernel of the ROCKEY4 SMART dongle is a smart card chip. The dongle also contains a memory chip. The data on that memory chip will not be lost when the dongle is powered off. The memory is divided into a user memory area, a module area, an algorithm area, and a user ID area. The developer can store important information of his software (e.g. the serial number) in the dongle. Note that, the number of the write cycles for each memory unit is 500,000 to 1,000,000, while the number of the read operation is not restricted. The number of 500,000 is a pretty large number, and can completely meet most requirements of developers. The dedicated smart card chip contains a random number generator, a seed code generator, and a user-defined algorithm interpreter, etc.

2.2 Hardware Interface

The ROCKEY4 SMART dongle supports USB 1.1 standard. With a USB hub, you can connect a maximum number of 16 dongles to one computer at one time. Each dongle is equipped with an indicator for informing some common problems. (Normally the indicator is always on after the dongle is plugged into the USB port. Otherwise, the hardware has a problem or the driver is not loaded.)

2.3 Considerations

Theoretically, the ROCKEY4 SMART dongle is a plug-and-play device, and can be plugged in and removed at any time. However, instability may be caused if it is removed when being accessed by a program.

Chapter 3 Development Kit

You can find the development package on the CD of the development kit (the box), and then encrypt your software with the tools and interfaces of the package. To enable you to install the contents of the CD on your computer, a file Setup.exe, which is located under the root directory, is provided.

3.1 Contents of the CD

The contents of this CS are introduced in the following two sections.

3.1.1 Tools

- 1) Ry4S_Editor.exe: It is an editing and testing tool located under directory “Tools\Editor”. You can use it to edit the contents of the dongle, test the dongle, test the contents you have written to the dongle, write to a set of dongles and more. All functions of the dongle can be utilized with this tool.
 - 2) RyEnvX.exe: It is an envelope encryption tool located under directory “Tools\Envelop”. You can use it to encrypt your software without programming. It can be used for encrypting more than one file at a time.
 - 3) .Net program encryption tool: It is located under directory “Tools\NetShell”, used for encrypting programs developed with C#, VB.NET, Managed C++ etc. It also provides a timing license function. Flash encryption tool: is provided under directory “Tools\SwfEnv”. By playing an encrypted .swf file using a flash player with an envelope, the copyright interest of the developer is maintained.
- For more information on how to use the envelope encryption tool, the .NET program encryption tool, and the flash encryption tool, see *ROCKEY4 SMART License Management*.

3.1.2 Development Interfaces

The interfaces are distinguished with different directory names. For example, the Delphi interfaces are located under directory Delphi. For Windows developers, the DLL interface is

provided. The calling of the DLL is supported for most of Windows development environments.

3.2 Installing Development Kit

To install the development kit of the ROCKEY4 SMART dongle, do as follows:

Step 1

This is a welcome page. Please close other programs before installation, as shown in Figure 3-1.

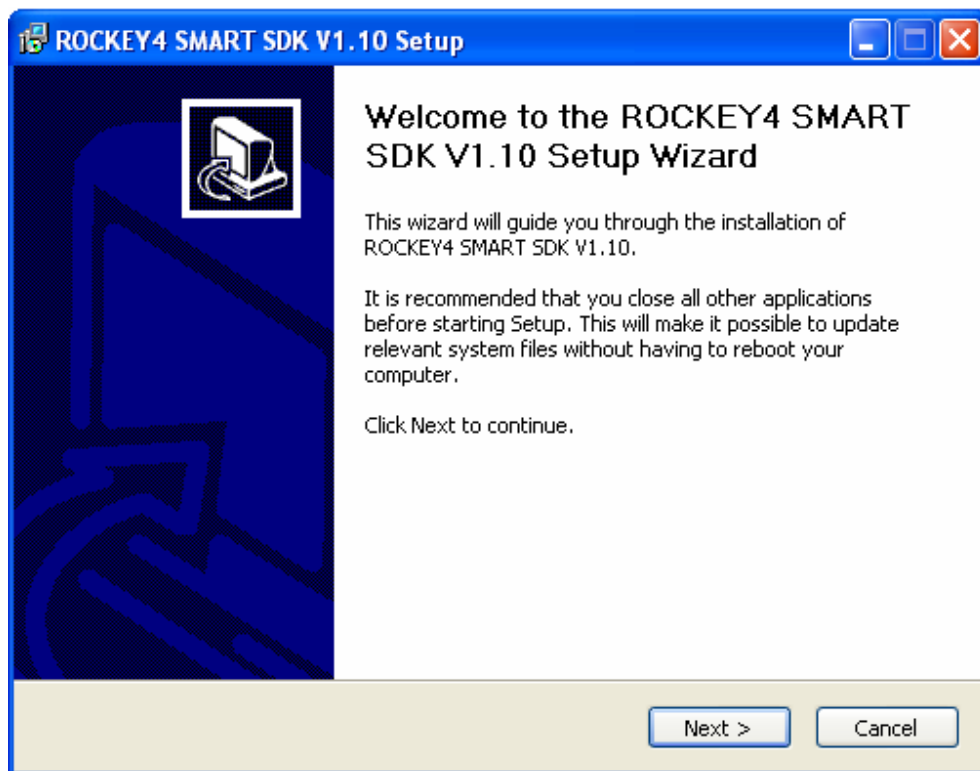


Figure 3-1 Welcome Screen

Step 2

This is the license statement. To proceed to the next step, choose "I Agree":

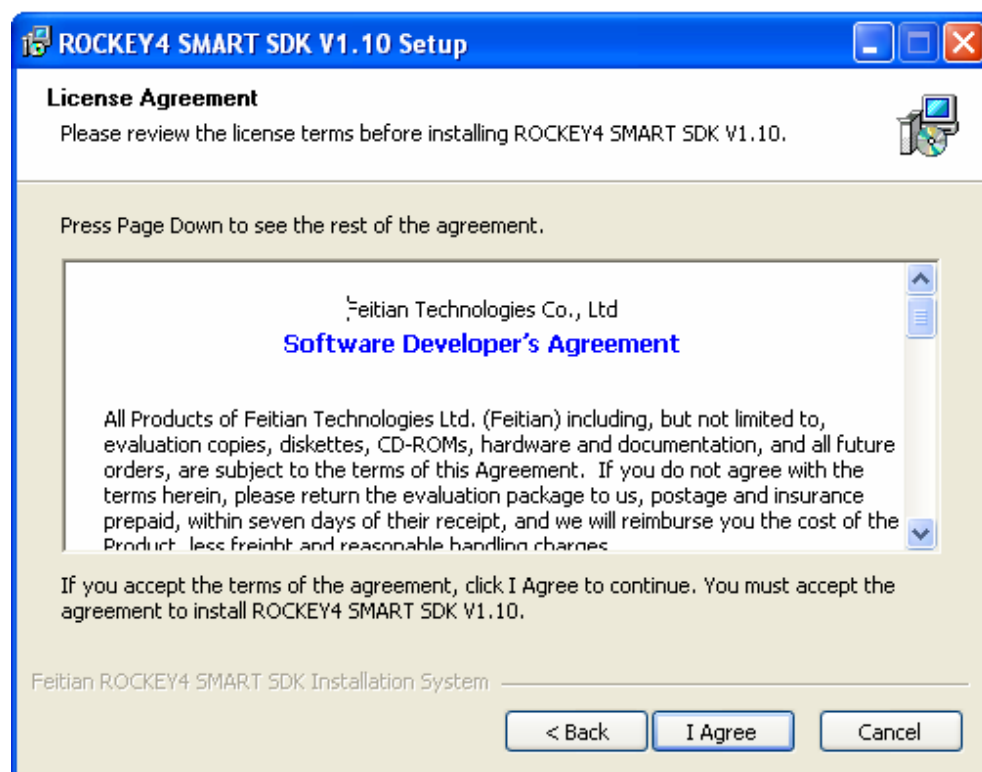


Figure 3-2 License Agreement

Step 3

Choose installation mode and installed components, as shown in Figure 3-3.

If you choose "Typical", the installer will install ROCKEY4 SMART application program interface libraries, application program interface header files, help documents, encryption tools, and samples.

If you choose "Compact", the installer will install ROCKEY4 SMART application program interface libraries and application program interface header files only.

If you choose "All", the installer will install all development components and development manuals of ROCKEY4 SMART.

If you choose "Custom", you can select the components you want to install.

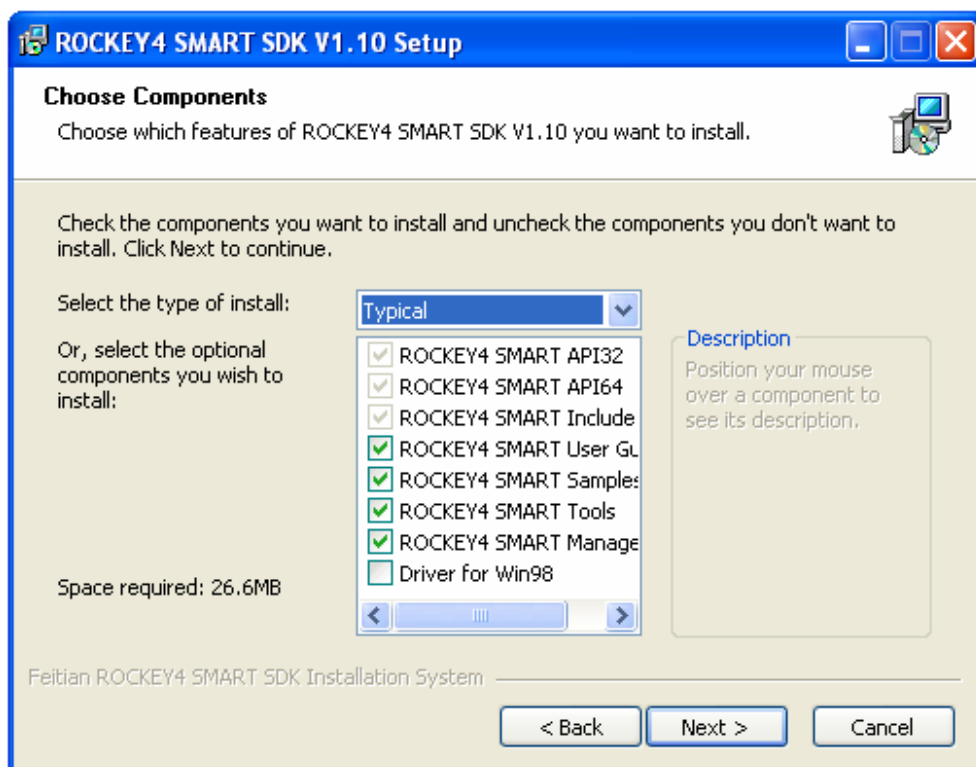


Figure 3-3 Installation Mode Selection

Step 4

Choose an installation path, as shown in Figure 3-4:

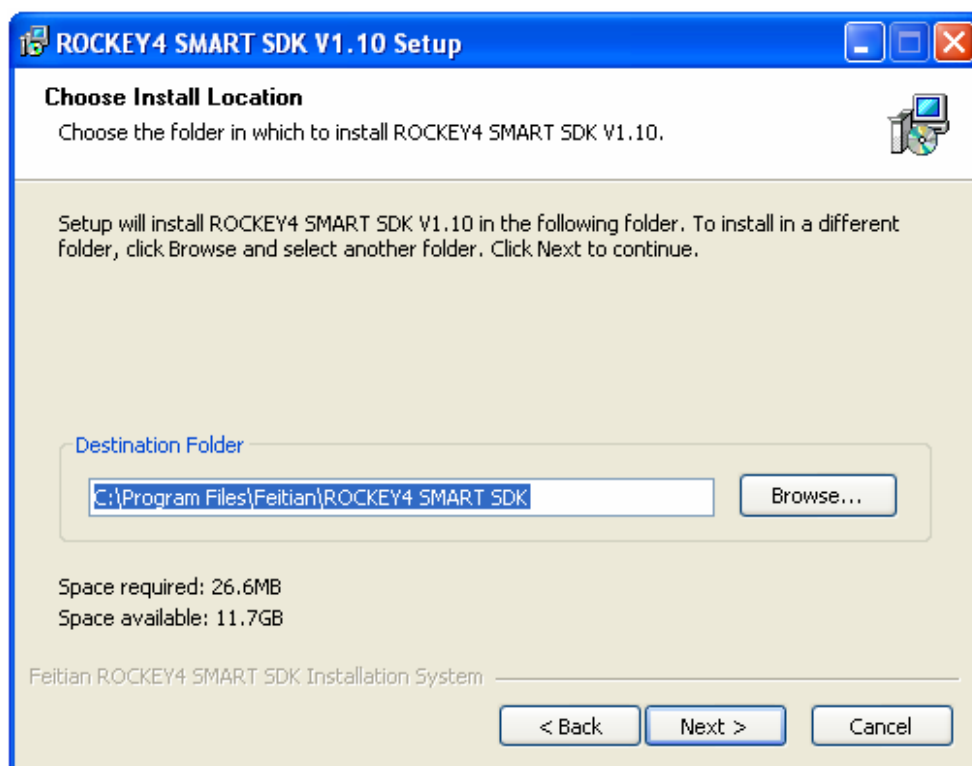


Figure 3-4 Installation Path Selection

Step 5

When the installation is completed normally, the following window will appear, as shown in Figure 3-5:

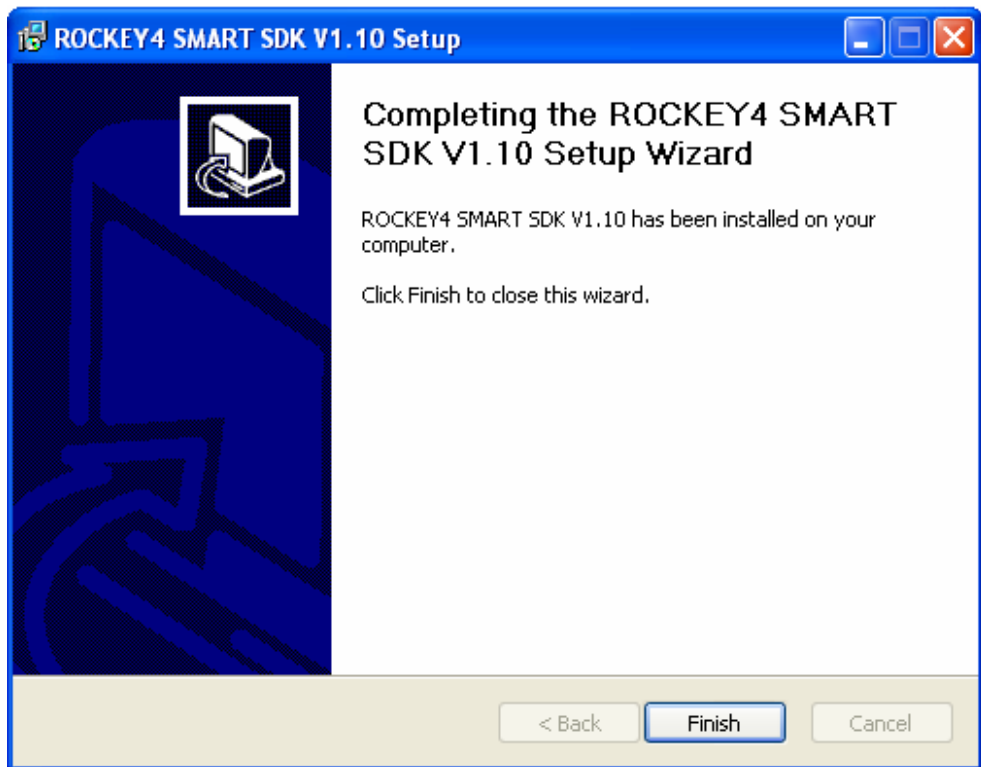


Figure 3-5 Completing Installation

3.3 Uninstalling Development Kit

To uninstall the plugin, you can:

- 1) Delete "ROCKEY4 SMART" from "Add or Remove Programs" in "Control Panel"; or
- 2) Select "Start" → "Programs" → "Rockey4 Smart SDK V1.10", and choose "Uninstall ROCKEY4 SMART SDK V1.10".

3.4 Driver Installation under Windows 98 SE

If you are using Windows 98 SE, when you plug a ROCKEY4 SMART dongle into your computer, you may be asked to insert the Windows 98 SE installation CD. Please insert the

ROCKEY4 SMART installation CD and change the directory to “Driver For Win98” and install it.

Chapter 4 Basic Concepts

To protect your software using the ROCKEY4 SMART dongle, you should read this chapter carefully to understand the basic concepts of the dongle.

4.1 Passwords

Each developer that orders the ROCKEY4 SMART dongle will be provided with 4 passwords, each has a length of 16 bits. Two of them are basic passwords (or first-level passwords). The others are advanced passwords (or second-level passwords). For example, the public passwords for the evaluation kit are C44C (P1), C8F8 (P2), 0799 (P3), and C43B (P4). The first two passwords are the first-level passwords. The other two passwords are the second-level passwords. These passwords are written before the dongle is delivered, and can be modified by developers. To gain all rights to access the dongle, developers must enter all of the passwords properly. The advanced passwords should be excluded from the delivery to end users, because the basic passwords are sufficient for them. **At a point in a program where the passwords should be referenced, both of the advanced passwords must be set to 0.** The following sections will explain which passwords are needed in a specific situation.

4.2 Hardware ID

Each ROCKEY4 SMART dongle contains a globally unique hardware ID. This ID is written to it by the manufacturer and even the manufacturer cannot change it later. With this ID, the unique validity can be validated by developers when the encryption is conducted for a particular user.

Operational attribute:

Readable with both first and second level passwords; not writable with both first and second

level passwords.

4.3 User Memory Area

Developers can read or write to this area. For example, you can write some important information required for the software to this area to check if the software that is running is valid or not. Thus, the software works only if the dongle is coupled to it.

This area is divided into two parts: low address section (0 - 499) and high address section (500 – 2000).

Operational attribute:

For low address section, readable and writable with both first and second level passwords;

For high address section, readable with first level passwords, readable and writable with second level passwords.

For version earlier than 1.03, RY_READ and RY_WRITE can be used to read or write to the dongle. For version 1.03 or later, RY_READ_EX and RY_WRITE_EX are recommended.

4.4 Module Characters

Module characters are stored in special units of the dongle designed to implement multi-module encryption. The units can also be used to store particular data for software encryption. Their usage depends on whether the envelope encryption or the API calling is used.

64 units are reserved in the ROCKEY4 SMART dongle for storing module characters. The length of each unit is 16 bits. In other words, 64 modules can be encrypted at a time. Developers can write to these units. A module is available only if the module character content is not 0. End users can confirm if a module is available or not by checking the module character property. If they want to know the content of the module character exactly, they must read the content using a self-defined algorithm. Therefore, the security is increased.

Operational attribute:

Not readable with first and second level passwords; writable with only second level passwords.

4.5 Module Property Characters

The property characters consist of 8 units, each with a length of 16 bits. The first 8-bit character is the module availability character, and the last 8-bit character is the module decreasability character.

Each bit of the module availability property character indicates if a corresponding module character is 0 or not.

Each bit of a module decreasability property indicates if a corresponding module character is decreasable. 1 stands for Yes. 0 stands for NO.

Operational attribute:

For module availability property: readable with both first and second level passwords; not writable with both first and second level passwords.

For module decreasability property: readable with both first and second level passwords; writable with only second level passwords.

4.6 Algorithm Area

This area is used for storing the self-defined algorithm written by developers. It includes 128 units, each with a length of 16 bits. Each instruction occupies a unit. Thus, developers are allowed to define an algorithm with 128 instructions (for details see Chapter 7).

Operational attribute:

Not readable with first and second level passwords; writable with second level passwords.

4.7 User ID

It is a serial number unit for managing the published software by developers. The ID is a 32-bit number stored at a particular location inside the dongle. Of course, developers are allowed to write other information to the location, such as a time value or product management related information etc.

Operational attribute:

Readable with both first and second level passwords; writable with only second level passwords.

4.8 Random Number

A random number can be generated with the ROCKEY4 SMART dongle. The random number can be used for anti-tracing, or as an operation factor in a hardware algorithm etc.

4.9 Seed Code and Return Codes

By providing a seed code to the ROCKEY4 SMART dongle, you can obtain 4 return codes for this seed code through an internal algorithm operation. The seed code algorithm is not public. Dongles with the same passwords generate identical return codes when the same seed code is input. For dongles with different passwords, different return codes will be generated even if the same seed code is input. By verifying this dependency, you can determine if the dongle is the expected one or not.

4.10 Timer and Counter

In current version of SDK, the functions of timer and counter are enhanced. 64 timer units and 64 counter units are available for controlling the use of programs in terms of time and number of uses respectively. In the timer unit, you can write a number of hours or an expiration date. By designing the software to access one of the 64 timer units, the timing license is implemented. Similarly, the number of uses can also be controlled using the counter units.

If a number of hours is written to a timer unit, the time value will be decreased automatically once the dongle is connected to the computer, without the need of other means of control.

For information on how to call the functions see Articles 32 to 36 in Section 6.2.

4.11 Dongle Configuration Update

In the versions 1.03 and above of SDK, a function of updating the overall configuration of the dongle is available. On the premise of key being verified, all configuration of the dongle is

updated at a time by calling the API interface, including the user ID, the memory area, the module value, the algorithm area, and the time and number counter units.

The detailed information about how to use the tools provided by the SDK and use the API function interface to update the dongle is available in *Rockey4 Smart License Management*.

Chapter 5 Dongle Editor

5.1 Introduction

The ROCKEY4 SMART Editor is a tool for performing operations, such as modifying, testing, batch testing, and batch reading and writing, on the ROCKEY4 SMART dongle. This tool is located at directory “Tools” under the installation directory or on the CD. The interface of the Editor (Ry4S_Editor.exe) includes 5 parts: the toolbar and dropdown menu, the status bar, the tree view, the operation status record, and the operation main window (see Figure 5-1).

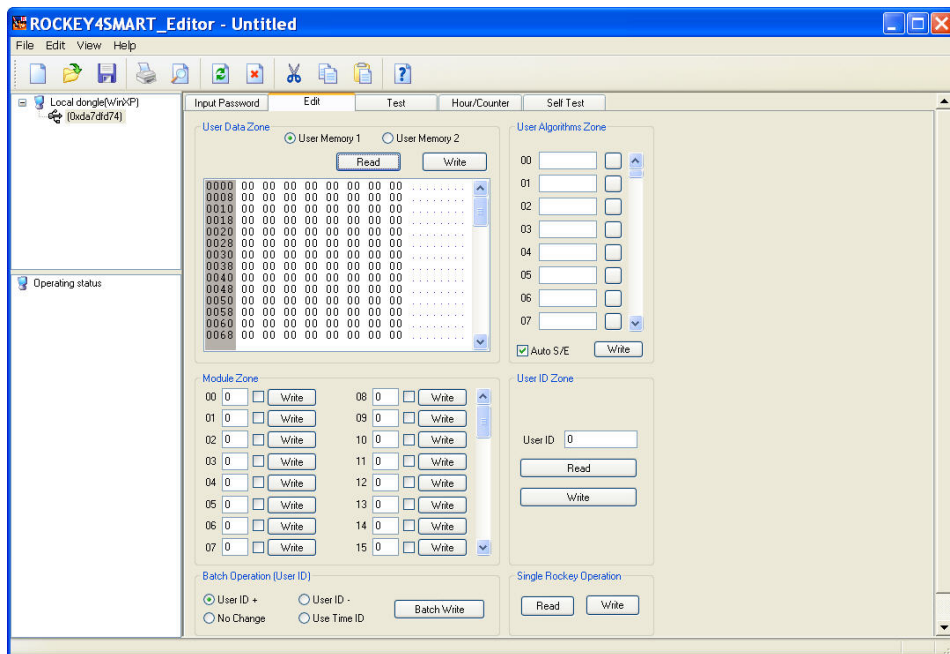


Figure 5-1

- (1) Toolbar and dropdown menu: They are located on the top of the window. Some functions,

such as print, save and refresh, can be used through activating the icons or the dropdown menu. A set of shortcut keys and icon buttons are also available there.

- (2) The status bar is located at the bottom of the window. It is used for indicating the working status of the ROCKEY4 SMART dongle, and the operation progress through a progress bar. The progress bar is not displayed when no operation is performed, or the operation has been finished. See Figure 5-2.

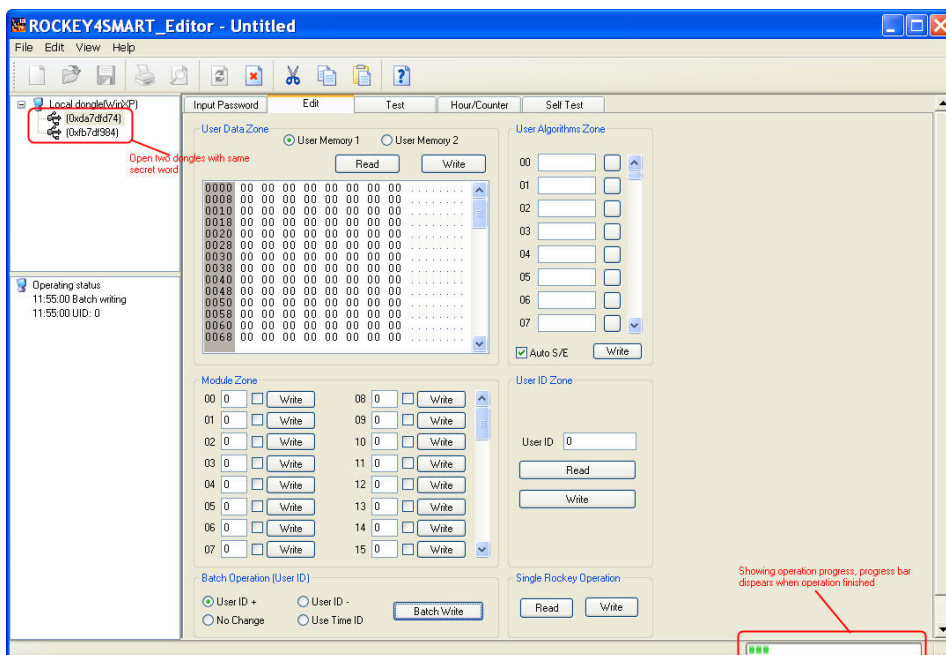


Figure 5-2

- (3) The tree view is located on the top left corner of the window. It displays the version of current operating system and the ROCKEY4 SMART dongles connected to the computer with same passwords. You can edit or test a particular dongle by selecting it from the list. The tree view can be refreshed using the menu item or the button on the toolbar.
- (4) The operation status column records the time, result, or error prompts (if any) of all operations. The list can be cleared using the menu item or the button on the toolbar. See Figure 5-3.

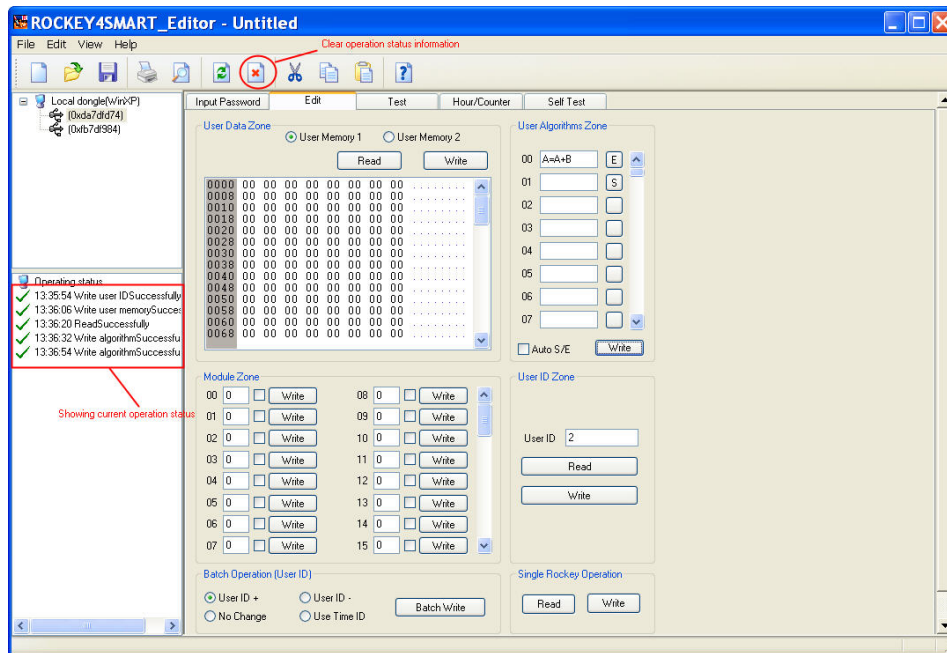


Figure 5-3

- (5) The operation main window involves Input Password, Edit, Test, Self Test etc. Also, you can use the editing template to operate the ROCKEY4 SMART dongle. The template can be saved to the disk, or be loaded from the disk. Thus, it is easy for developers to initialize the ROCKEY4 SMART dongle.

The editor supports drag-and-drop and file relationship. By clicking on the template file of the explorer, the editor will start and you can edit the file then. The template file can be opened through the menu or using the explorer. In addition, the content of the template can be printed and previewed. Without the dongle, the editor is still available. The template file can be used with one or more dongles. During operating the dongle, you can make operations on the interface. The progress can be viewed from the progress bar.

Note:

1. Numerical Input/Display

Decimal representation is required for the number of seeds, the number of hours, and the

number of uses. Other numbers should be input and displayed in hex.

2. Do not edit the part of the template which is being operated when making an operation on the dongle. For example, the data of the memory part of the template must not be edited when writing to the memory.

5.2 Description of Operations

5.2.1 Entering Passwords

First, you must enter the basic passwords and the advanced passwords of your ROCKEY4 SMART dongle. See Figure 5-4.

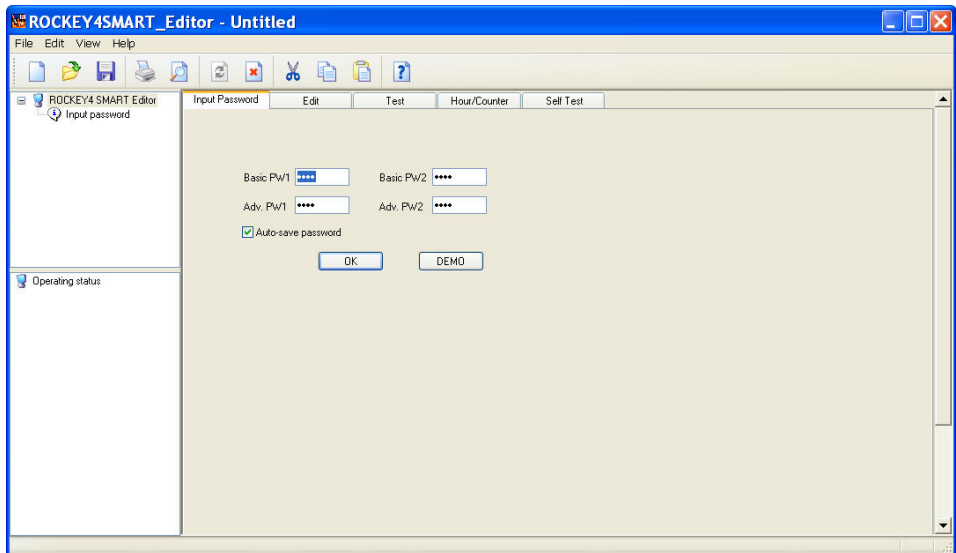


Figure 5-4

Be sure to type correct passwords. If the basic passwords are incorrect, the dongle cannot be used normally. If the basic passwords are correct, but the advanced passwords are not correct, a problem may occur when writing to the dongle, though the dongle can be found and read.

To perform an operation on the DEMO dongle, click DEMO button. The public passwords are C44C (P1), C8F8 (P2), 0799 (P3), and C43B (P4).

If you have selected Auto-save password option, the passwords will be encrypted and saved

automatically, so that the action of entering them again and the possibility of entering errors can be both avoided. After entering the passwords, you are logged into the ROCKEY4 SMART Editor, and the corresponding dongles with the same passwords will be automatically searched.

5.2.2 Editing

Edit the selected ROCKEY4 SMART dongle. Now you can see the only identity of this ROCKEY4 SMART dongle, i.e., hardware ID. Even the manufacturer of this dongle cannot change the hardware ID of the dongle.

As shown in Figure 5-5, the interface includes 6 areas: user memory area, module area, algorithm area, user ID area, single dongle operation area, and batch operation area.

In the user memory area, you can read data from or write data to the dongle. The data may be in hex or ASCII format. After you click Read or Write button, a progress bar will be displayed in the status bar. When the operation is complete, the result will be shown in the status bar.

You can perform an operation on a particular module in the module area. You can input a value and/or specify if decrease is allowed.

In the algorithm area, you can write a set of algorithm statements to the dongle. An algorithm statement consists of operands and operators (e.g. $A=A+B$). (For more information on the algorithm, see Chapter 7). On the right side of each statement, there is a button, which indicates if the statement is the starting (S), middle (blank), ending (E), or standalone (SE) statement. If you click on a button, the indication flag will appear repeatedly. If you have selected "Auto S/E" option, the indication flags like the starting or the ending flags will be added according to the algorithm intervals automatically, as shown in Figure 5-5.

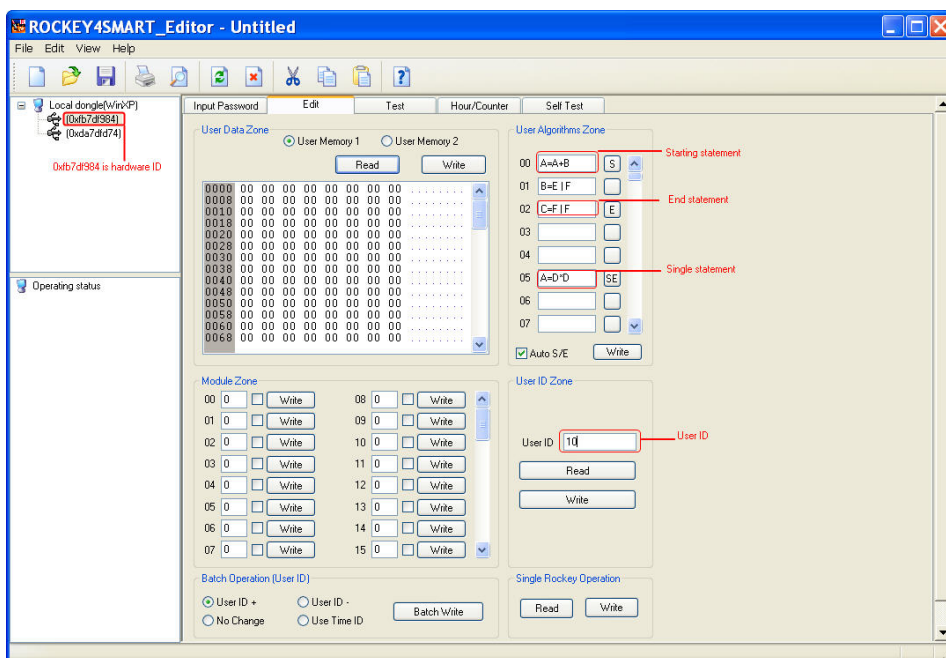


Figure 5-5

You can read a user-defined ID from or write a user-defined ID to the dongle in the “User ID Zone”.

In the “Batch Operation” area, you can write to a batch of dongles. Type a starting ID in the “User ID” field. Select a manner in which the IDs will be set (see below). Click “Batch Write” button. You will write data to a batch of dongles. After that, the data on these dongles is identical, except that the user ID may be different (depending on the manner in which the ID is generated as you specified in the following ways).

- 1) User ID +: The value entered into the User ID field will be written to the first dongle in the device selection area. After being increased by 1, the value is written to the next dongle, and so forth. If you initially specify a user ID “10”, the next user ID written to the second dongle will be “11”.
- 2) User ID -: The user ID will be decreased by 1.
- 3) Use Time ID: Use the system time as user ID.

- 4) No Change: The value in the User ID field will be written to all dongles in the device selection area without any changes.

In the single dongle operation area, you can perform an operation on a single dongle you select at a time.

Note: The value of a module and the algorithm cannot be read. You can only write them.

5.2.3 Testing

Test the selected ROCKEY4 SMART dongle(s). The testing interface includes the user memory area, the algorithm area, the user ID area, the module property area, and the seed calculation area, as shown in Figure 5-6.

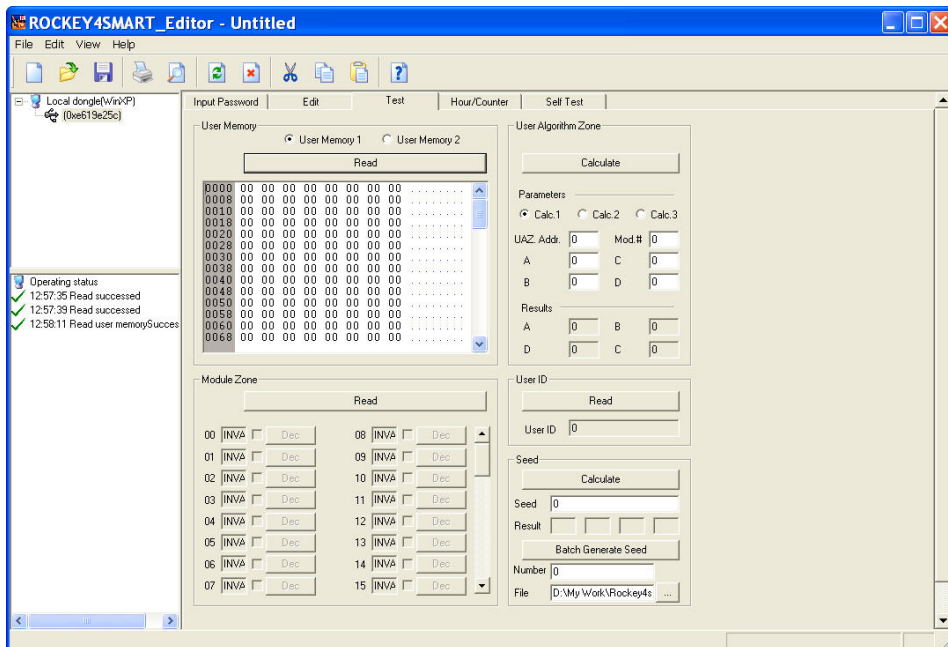


Figure 5-6

- 1) The user memory is defined by developers. Data can be displayed in hex or ASCII format. To read data from the user memory, click “Read” button.
- 2) In the user algorithm area, choose an algorithm you want to test. If you choose “Calc.

1” or “Calc. 3”, a module number input box will appear. If you choose Calc. 2, a seed input box will appear. Type the initial address of the algorithm and the values of parameters A, B, C, and D in hex format. Enter the module number or seed. Click “Calculate”. The results will be displayed in “Results” section. The address refers to the location of the starting statement or the standalone statement.

- 3) In the user ID area, click “Read” to read a user-defined ID. The size of the user ID area on the dongle is 32 bits.
- 4) In the module property area, the validity and degression properties of a module are displayed. To refresh this area, click “Read” button. If “Dec” buttons are greyed out, the values of the modules cannot be decreased. Otherwise, you can decrease the value by 1 by clicking on “Dec” button.
- 5) The seed calculation area includes 2 sections. Four results of seed calculation are displayed in the top section. In “Number” field, enter a decimal number. The random seed and results will be written to a file. By default, the filename is “Random_Seed.txt” under the executable directory.

5.2.4 Timing and Number of Uses

There are 64 timer units and 64 counter units in the dongle. On “Hour/Counter” page, you can specify or view the values of these units. The upper half window is the values for timer units, while the lower half is the values of counter units.

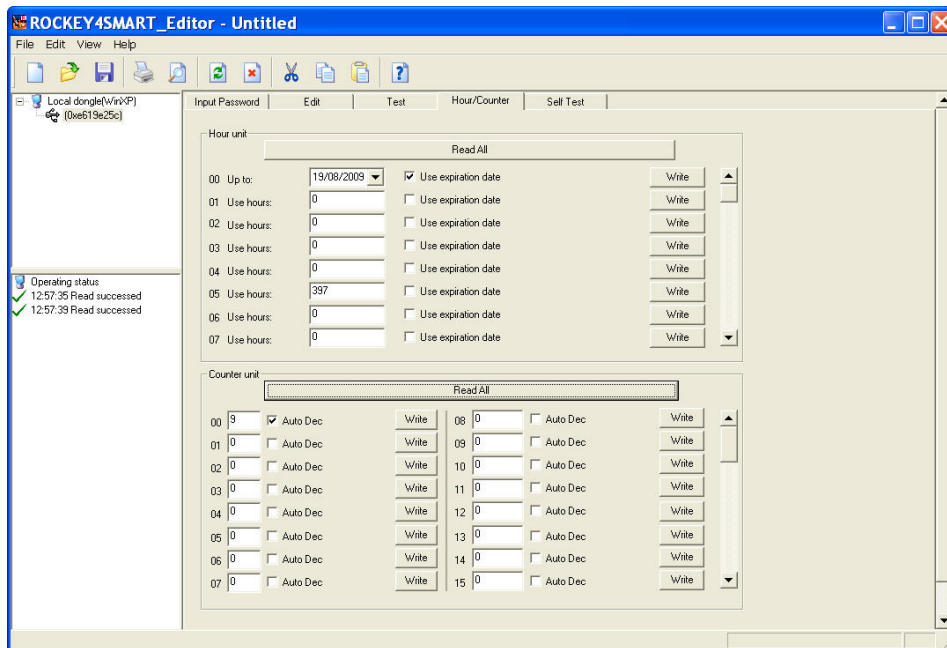


Figure 5-7

- 1) The value of a timer unit can be a number of hours remained or an expiration date. By checking “Use expiration date”, the timing license mode can be switched to use an expiration date. After specifying a value, click “Write” to save it to the dongle.
- 2) Each counter unit stores an integer value. By clicking “Write” button, the integer value in the text box will be written to the counter unit of the dongle. If you select “Auto Dec” option, the value of the counter unit will be decreased by 1 each time the program is executed, until the value equals to 0.

Note:

When using versions earlier than v1.03, it is recommended that both the number of hours in the timer unit and the number in the counter unit should not exceed 65534. Moreover, the expiration date in the timer unit should not exceed 2013. Otherwise, the result will not be accurate.

For version 1.03 or higher, the number of hours should be lower than 70000000 and

the value of the counter should be lower than 400000000 for the same reason.

5.2.5 Self-testing

You can perform a self-testing operation on all ROCKEY4 SMART dongles or on the selected ones, as shown in Figure 5-8.

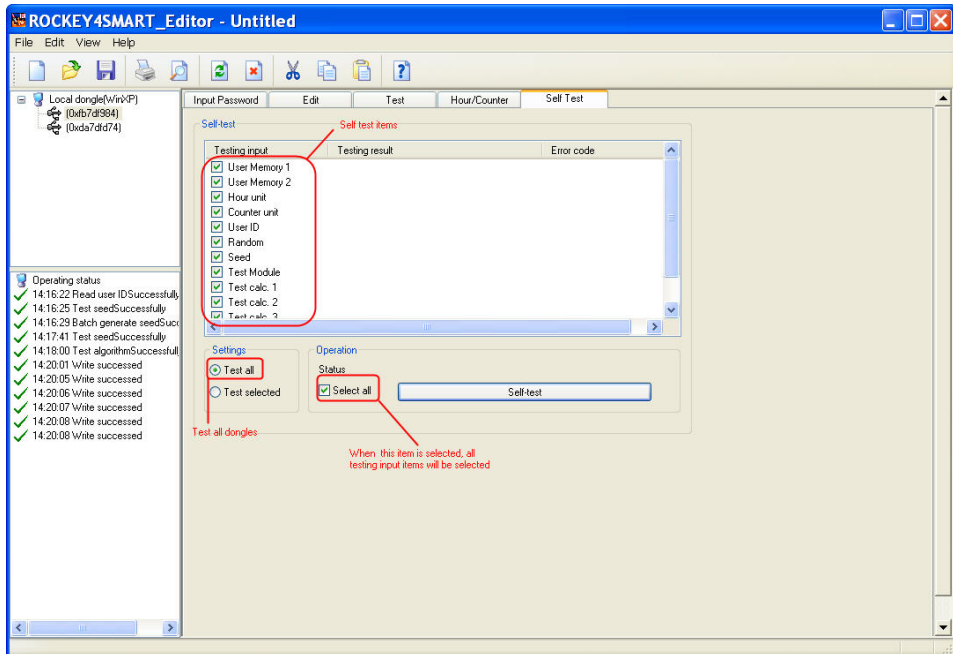


Figure 5-8

Important notes:

The content of the dongle will be destroyed by this operation. The data on the dongle will be cleared after performing the operation, except the password information.

For version 1.03 or higher, the value of the time unit will be the current date after the self-testing operation. But the value is 0 hour after the operation for other versions.

5.3 Save your Work

The template can be saved to the disk or be printed to back up, as shown in Figure 5-9.



Chapter 6 Calling API Functions

The Application Programming Interface (API) of the ROCKEY4 SMART dongle is the most basic, as well as the most flexible encryption way. You can make the most out of the dongle by using the API to encrypt a program. The effectiveness of encryption depends on the way you use the API. The more complex the API calling is, the better the software is protected.. The API interface has been simplified to the maximum level.

Applications connect to the ROCKEY4 SMART dongle through the API. You can place the points for checking the presence of the dongle anywhere in the applications and determine how to respond to the result of checking. Alternatively, you can check the data in the user memory area of the dongle in your applications.

If you want to encrypt a program with the API, you can use the Editor to perform some necessary operations first, such as setting modules, writing to user memory, writing algorithms and user ID, etc. Of course, these operations can also be done by using the API.

Below, we use examples with the C interface to illustrate the meaning of the interface parameters. Similarly, you may call other language interfaces in the same way.

6.1 Function Prototype and Definition

WORD Rockey

```
(  
    WORD    function,  
    WORD*   handle,  
    DWORD*  lp1,  
    DWORD*  lp2,  
    WORD*   p1,
```

```
WORD*  p2,  
WORD*  p3,  
WORD*  p4,  
BYTE*  buffer
```

```
);
```

There is only one function for the ROCKEY4 SMART dongle. All functions of the dongle can be used by calling this interface. This function has multiple functions in itself.

The following is an example of C language calling. The following descriptions are all based on this example.

```
retcode = Rockey(function, handle, lp1, lp2, p1, p2, p3, p4, buffer);
```

Definition of parameters of ROCKEY function:

Name	Type	Meaning
Function	16-bit function number	API function
Handle	Pointer to 16-bit handle	Pointer to the operation handle
lp1	Pointer to 32-bit integer	Long parameter 1
lp2	Pointer to 32-bit integer	Long parameter 2
p1	Pointer to 16-bit integer	Parameter 1
p2	Pointer to 16-bit integer	Parameter 2
p3	Pointer to 16-bit integer	Parameter 3
p4	Pointer to 16-bit integer	Parameter 4
Buffer	Pointer to 8-bit integer	Pointer to character buffer

Note:

All interface parameters must be defined in the program. Passing a NULL pointer is not allowed.

1) **function** is a 16-bit number, which represents the specific function. The meaning of the function is defined as follows:

RY_FIND	1	Find dongle
RY_FIND_NEXT	2	Find next dongle
RY_OPEN	3	Open dongle
RY_CLOSE	4	Close dongle
RY_RANDOM	7	Generate random number
RY_SEED	8	Generate seed code
RY_WRITE_USERID	9	Write user ID
RY_READ_USERID	10	Read user ID

RY_SET_MODULE	11	Set module character
RY_CHECK_MODULE	12	Check module status
RY_WRITE_ARITHMETIC	13	Write algorithm
RY_CALCULATE1	14	Calculation 1
RY_CALCULATE2	15	Calculation 2
RY_CALCULATE3	16	Calculation 3
RY_DECREASE	17	Decrease module unit
RY_SET_RSAKEY_N	29	Set RSA private key – N; the public key is 65537
RY_SET_RSAKEY_D	30	Set RSA private key – D; the public key is 65537
RY_SET_DES_KEY	41	Set DES key
RY_DES_ENC	42	DES/3DES encryption
RY_DES_DEC	43	DES/3DES decryption
RY_RSA_ENC	44	RSA encryption
RY_RSA_DEC	45	RSA decryption

The following is the functions of Read/Write, Count time, Count number, and Update used in versions earlier than v1.03. These functions are not recommended. They are reserved and listed here for compliance purpose only.

RY_READ	5	Read dongle
RY_WRITE	6	Write to dongle
RY_SET_COUNTER	20	Set the value of a number counting unit
RY_GET_COUNTER	22	Read the value of a number counting unit
RY_DEC_COUNTER	23	Decrease the value of number counter by 1
RY_SET_TIMER	24	Set the value of a clock unit
RY_GET_TIMER	25	Read the value of a clock unit
RY_ADJUST_TIMER	26	Adjust the clock inside the dongle

The following is the enhanced functions of version v1.03 or above. These functions are recommended, but they cannot be used together with the old functions listed above.

RY_READ_EX	46	Read memory
RY_WRITE_EX	47	Write to memory
RY_SET_COUNTER_EX	160	Set the value of a number counting unit
RY_GET_COUNTER_EX	161	Read the value of a number counting unit
RY_SET_TIMER_EX	162	Set the value of a clock unit
RY_GET_TIMER_EX	163	Read the value of a clock unit
RY_ADJUST_TIMER_EX	164	Adjust the clock inside the dongle
RY_UPDATE_GEN_EX	166	Generate the content of an update file
RY_UPDATE_EX	168	Update
RY_SET_UPDATE_KEY	169	Set update key-pair
RY_ADD_UPDATE_HEADER	170	Fill license file header
RY_ADD_UPDATE_CONTENT	171	Fill the content of license file
RY_GET_TIME_DWORD	172	Convert time (DWORD type)

2) **handle** is the pointer to the operation handle.

3) **lp1** and **lp2** are pointers to long integer parameters. Their contents depend on the specific function.

4) **p1**, **p2**, **p3**, and **p4** are pointers to short integer parameters. Their contents depend on the specific function.

6) **buffer** is a pointer to the character buffer. Its content depends on the specific function.

6.2 ROCKEY4 SMART API Services

The API services are described below in detail. The functions marked with [*] require both of the two advanced passwords (p3, p4).

Note: p3 and p4 are advanced passwords. They are used for developers to operate the dongle only, and should not appear in the software provided to end users. These passwords should be set to “0” when searching for dongles in the software provided to end users.

1. RY_FIND

For: Check if the dongle with specified passwords exists

Input Parameters:

function = RY_FIND

*p1 = password 1

*p2 = password 2

*p3 = password 3 (optional)

*p4 = password 4 (optional)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful, the hardware ID of the dongle will be written to *lp1.

2. RY_FIND_NEXT

For: Check if the dongle with specified passwords still exists

Input Parameters:

function = RY_FIND_NEXT

*p1 = password 1

*p2 = password 2

*p3 = password 3 (optional)

*p4 = password 4 (optional)

*lp1 = the hardware ID of the last dongle found by RY_FIND or RY_FIND_NEXT

Return Values:

If the operation is successful, the hardware ID of the dongle will be written to *lp1.

3. RY_OPEN

For: Open the dongle with specified passwords and hardware ID

Input Parameters:

function = RY_OPEN

*p1 = password 1

*p2 = password 2

*p3 = password 3 (optional)

*p4 = password 4 (optional)

*lp1 = hardware ID

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful, the handle of the dongle is written to *handle.

4. RY_CLOSE

For: Close the dongle with specified handle

Input Parameters:

function = RY_CLOSE

*handle = handle of the dongle

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

5. RY_READ

For: Read the user memory of the dongle

Input Parameters:

function = RY_READ

*handle = handle of the dongle

*p1 = location

*p2 = length

buffer = pointer to the buffer

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful, the content of the memory will be written to the buffer.

6. RY_WRITE

For: Write content to user memory

Input Parameters:

function = RY_WRITE

*handle = handle of the dongle

*p1 = location

*p2 = length

buffer = pointer to the buffer

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

7. RY_RANDOM

For: Get a random number from the dongle

Input Parameters:

function = RY_RANDOM

*handle = handle of the dongle

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful, the random number is carried by *p1, *p2, *p3, and *p4.

8. RY_SEED

For: Get the return code of the seed

Input Parameters:

function = RY_SEED

*handle = handle of the dongle

*lp2 = seed

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful:

*p1 = return code 1

*p2 = return code 2

*p3 = return code 3

*p4 = return code 4

9. [*] RY_WRITE_USERID

For: Write user-defined ID

Input Parameters:

function = RY_WRITE_USERID

*handle = handle of the dongle

*lp1 = user ID

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

10. RY_READ_USERID

For: Read user-defined ID

Input Parameters:

function = RY_READ_USERID

*handle = handle of the dongle

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful, the user ID is carried by *lp1.

11. [*] RY_SET_MODULE

For: Set a module character and degression property

Input Parameters:

function = RY_SET_MODULE

*handle = handle of the dongle

*p1 = module number

*p2 = user module character

*p3 = if degression is allowed (1 = yes; 0 = no)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

12. RY_CHECK_MODULE

For: Check module property character

Input Parameters:

function = RY_CHECK_MODULE

*handle = handle of the dongle

*p1 = module number

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful:

*p2 = 1 indicates that the module is valid

*p3 = 1 indicates that the module can be decreased

13. [*] RY_WRITE_ARITHMETIC

For: Write a self-defined algorithm to the dongle

Input Parameters:

function = RY_WRITE_ARITHMETIC

*handle = handle of the dongle

*p1 = location of algorithm area

buffer = string of instructions of algorithm

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

14. RY_CALCULATE1

For: Let the dongle perform an operation in the specified way; the result depends on the user algorithm

Input Parameters:

function = RY_CALCULATE1

*handle = handle of the dongle

*lp1 = start point of operation

*lp2 = module number

*p1 = input value 1

*p2 = input value 2

*p3 = input value 3

*p4 = input value 4

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful:

*p1 = return value 1

*p2 = return value 2

*p3 = return value 3

*p4 = return value 4

15. RY_CALCULATE2

For: Let the dongle perform an operation in the specified way; the result depends on the user algorithm

Input Parameters:

function = RY_CALCULATE2

*handle = handle of the dongle

*lp1 = start point of operation

*lp2 = seed

*p1 = input value 1

*p2 = input value 2

*p3 = input value 3

*p4 = input value 4

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful:

*p1 = return value 1

*p2 = return value 2

*p3 = return value 3

*p4 = return value 4

16. RY_CALCULATE3

For: Let the dongle perform an operation in the specified way; the result depends on the user algorithm

Input Parameters:

function = RY_CALCULATE3

*handle = handle of the dongle

*lp1 = start point of operation

*lp2 = start address of module character

*p1 = input value 1

*p2 = input value 2

*p3 = input value 3

*p4 = input value 4

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful:

*p1 = return value 1

*p2 = return value 2

*p3 = return value 3

*p4 = return value 4

17. RY_DECREASE

For: Perform a decrease operation on the specified module character

Input Parameters:

function = RY_DECREASE

*handle = handle of the dongle

*p1 = module number

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

18. [*] RY_SET_DES_KEY

For: Set DES key

Input Parameters:

function = RY_SET_DES_KEY

*handle = handle of the dongle

*p1 = algorithm selection: 0 – DES; 1 - 3DES

Buffer = 8 or 16-byte key

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

19. RY_DES_ENC

For: DES/3DES encryption

Input Parameters:

function = RY_DES_ENC

*handle = handle of the dongle

*p1 = algorithm selection: 0 – DES; 1 - 3DES

*p2 = length of the data to be encrypted; must be a multiple of 8

Buffer = data to be encrypted; maximum length is 1024 bytes each time

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

20. RY_DES_DEC

For: DES/3DES decryption

Input Parameters:

function = RY_DES_DEC

*handle = handle of the dongle

*p1 = algorithm selection: 0 – DES; 1 - 3DES

*p2 = length of data to be decrypted; must be a multiple of 8

Buffer = data to be decrypted; maximum length is 1024 bytes each time

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

21. [*] RY_SET_RSAKEY_N

For: Set RSA private key – N; the public key is 65537

Input Parameters:

function = RY_SET_RSAKEY_N

*handle = handle of the dongle

Buffer= RSA key N (0 - 127)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

22. [*] RY_SET_RSAKEY_D

For: Set RSA private key – D; the public key is 65537

Input Parameters:

function = RY_SET_RSAKEY_D

*handle = handle of the dongle

Buffer= RSA key D (0 - 127)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

23. RY_RSA_ENC

For: RSA encryption

Input Parameters:

function = RY_RSA_ENC

*handle = handle of the dongle

*p1 = 0 (encryption with private key) or 1 (encryption with public key)

*p2 = length of data to be encrypted (p3=0); Rockey padding type

*p3 = Rockey padding (P3=0); no padding (P3=1)

Buffer = input parameter; maximum length is 120 bytes each time; 128 bytes returned
(cipher-text data)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

24. RY_RSA_DEC

For: RSA decryption

Input Parameters:

function = RY_RSA_DEC

*handle = handle of the dongle

*p1 = 0 (decryption with private key) or 1 (decryption with public key)

*p2 = length of data to be decrypted (p3=0); Rockey padding type

*p3 = Rockey padding (P3=0); no padding (P3=1)

Buffer = input cipher-text data; length is 128 bytes; decrypted data returned

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

25. [*] RY_SET_COUNTER

For: Set the value of a counter unit

Input Parameters:

function = RY_SET_COUNTER

*handle = handle of the dongle

*p1 = number of the counter unit starting from 0

*P2 = value of counter (WORD type)

*P3 = 1 (degression allowed) or 0 (degression not allowed)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

26. RY_GET_COUNTER

For: Read the value of a counter unit

Input Parameters:

function = RY_GET_COUNTER

*handle = handle of the dongle

*p1 = number of the counter unit starting from 0

Output:

*P2 = value of counter (WORD type)

*P3 = 1 (degression allowed) or 0 (degression not allowed)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

27. RY_DEC_COUNTER

For: Decrease the value of a counter unit by 1

Input Parameters:

function = RY_DEC_COUNTER

*handle = handle of the dongle

*p1 = number of the unit

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

28. [*] RY_SET_TIMER

For: Set the value of a timer unit

Input Parameters:

function = RY_SET_TIMER

*handle = handle of the dongle

*p1 = number of the timer unit starting from 0

*p2 = number of hours

*p4 = 0 (hours) or 1 (date)

buffer = a date and time value of SYSTEMTIME type (expiration date)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

29. RY_GET_TIMER

For: Read the value of a timer unit

Input Parameters:

function = RY_GET_TIMER

*handle = handle of the dongle

*p1 = number of the timer unit starting from 0

Output:

*p2 = number of hours

*p4 = 0 (hours) or 1 (date)

buffer = a date and time value of SYSTEMTIME type (expiration date)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

30. RY_ADJUST_TIMER

For: Synchronize the dongle clock with the computer time if the clock is earlier than the computer time; do nothing if the clock is later than the computer time.

Input Parameters:

function = RY_ADJUST_TIMER

buffer = computer time of SYSTEMTIME type

Output:

buffer = If the function is called successfully, a time value of the dongle clock will be returned (SYSTEMTIME type)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

31. RY_READ_EX

For: Read user memory of the dongle (for version 1.03 or later)

Input Parameters:

function = RY_READ_EX

*handle = handle of the dongle

*p1 = location

*p2 = length

buffer = Pointer to buffer

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

If the operation is successful, the content of the memory will be read into buffer.

32. RY_WRITE_EX

For: Write to user memory (for version 1.03 or later)

Input Parameters:

function = RY_WRITE_EX

*handle = handle of the dongle

*p1 = location

*p2 = length

buffer = pointer to buffer

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

33. [*] RY_SET_COUNTER_EX

For: Set the value of a counter unit

Input Parameters:

function = RY_SET_COUNTER_EX

*handle = handle of the dongle

*p1 = number of the counter unit starting from 0

*P3 = 1 (degression allowed) or 0 (degression not allowed)

buffer[0~3] = value of the counter (DWORD type)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

34. RY_GET_COUNTER_EX

For: Read the value of a counter unit

Input Parameters:

function = RY_GET_COUNTER_EX

*handle = handle of the dongle

*p1 = number of the counter unit starting from 0

Output:

*P3 = 1 (degression allowed) or 0 (degression not allowed)

buffer[0~3] = value of the counter (DWORD type)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

35. [*] RY_SET_TIMER_EX

For: Set the value of a timer unit

Input Parameters:

function = RY_SET_TIMER_EX

*handle = handle of the dongle

*p1 = number of the timer unit starting from 0

*p3 = 1 (date) or 2 (hours)

buffer = a date and time value of SYSTEMTIME type (expiration date) or number of hours

(DWORD type)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

36. RY_GET_TIMER_EX

For: Read the value of a timer unit. Before reading, you need to synchronize the dongle clock with the computer time using RY_ADJUST_TIMER_EX function.

Input Parameters:

function = RY_GET_TIMER_EX

*handle = handle of the dongle

*p1 = number of the timer unit starting from 0

Output:

*p3 = 1 (date) or 2 (hours)

buffer = a date and time value of SYSTEMTIME type (expiration date) or number of hours

(DWORD type)

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

37. RY_ADJUST_TIMER_EX

For: Set the dongle clock to be the same as the computer if the clock is earlier than the computer time; otherwise, nothing will be done.

Input Parameters:

function = RY_ADJUST_TIMER_EX

buffer = computer time of SYSTEMTIME type

Return Values:

If retcode = 0, the operation is successful; other values indicate errors.

38. RY_GET_TIME_DWORD

For: Convert a date and time value to a number of minutes elapsed from 00:00 (hh:mm) of Jan. 1, 2006.

Input Parameters:

function = RY_GET_TIME_DWORD

*handle = handle of the dongle

*lp2 = year

*p1 = month

*p2 = day

*p3 = hour

*p4 = minute

Output:

*lp1 = number of minutes elapsed from 2006-1-1 00:00

If retcode = 0, the operation is successful; other values indicate errors.

39. For information on the interfaces related to update (RY_UPDATE_GEN_EX, RY_UPDATE_EX, RY_SET_UPDATE_KEY, RY_ADD_UPDATE_HEADER, RY_ADD_UPDATE_CONTENT), see *ROCKEY4 SMART License Management* and samples in the SDK.

6.3 Error Codes

ERR_SUCCESS	0	No error
ERR_NO_ROCKEY	3	No ROCKEY4 SMART dongle found
ERR_INVALID_PASSWORD	4	ROCKEY4 SMART dongle found, but basic password is not correct
ERR_INVALID_PASSWORD_OR_ID	5	Error password or hardware ID
ERR_SETID	6	Error in setting hardware ID

ERR_INVALID_ADDR_OR_SIZE	7	Error in reading/writing address or length
ERR_UNKNOWN_COMMAND	8	Invalid command
ERR_NOTBELEVEL3	9	Internal error
ERR_READ	10	Error in reading data
ERR_WRITE	11	Error in writing data
ERR_RANDOM	12	Error random number
ERR_SEED	13	Error seed code
ERR_CALCULATE	14	Error calculation
ERR_NO_OPEN	15	Dongle not opened before operation
ERR_OPEN_OVERFLOW	16	Too many dongles opened (>16)
ERR_NOMORE	17	No other dongle found
ERR_NEED_FIND	18	FindNext without Find once
ERR_DECREASE	19	Error degression
ERR_AR_BADCOMMAND	20	Error algorithm instruction
ERR_AR_UNKNOWN_OPCODE	21	Error algorithm operator
ERR_AR_WRONGBEGIN	22	The first instruction of the algorithm contains a constant
ERR_AR_WRONG_END	23	The last instruction of the algorithm contains a constant
ERR_AR_VALUEOVERFLOW	24	The value of the constant in the algorithm is greater than 63
ERR_TOOMUCHTHREAD	25	The number of threads that open the dongle in a process is greater than 100
ERR_RECEIVE_NULL	0x100	Cannot receive
ERR_UNKNOWN_SYSTEM	0x102	Unknown operating system
ERR_INVALID_RY4S	30	Attempt to operate a non-Rockey4 Smart dongle
ERR_SET_DES_KEY	40	Error setting DES key
RR_DES_ENCRYPT	41	DES encryption error
ERR_DES_DECRYPT	42	DES decryption error
ERR_SET_RSAKEY_N	43	Error setting N of RSA key
RR_SET_RSAKEY_D	44	Error setting D of RSA key
ERR_RSA_ENCRYPT	45	RSA encryption error
ERR_RSA_DECRYPT	46	RSA decryption error
ERR_INAVLID_LENGTH	47	Invalid length of data
ERR_UNKNOWN	0xffff	Unknown error

6.4 Basic Application Examples

Some program examples are provided to help beginners understand the use of the ROCKEY4 Smart dongle. These programs are intended to demonstrate some functions of the dongle only. To use the dongle in a smart way, you should make further development. Section 6.5 Advanced Application Examples is a good reference for some advanced encryption methods, but it

is only for reference. After all, the public experiences have no enough encryption strength to build applications with high security.

Some key points that you need to pay attention to when programming:

1. P3 and P4 are Advanced passwords enabling developers like you to write to the dongle when customizing the content of the dongle. They should always be set to zero in software delivered to end users.
2. Make sure that none of the parameters in the ROCKEY4 Smart functions is a Null pointer. For example, even if you do not require the Buffer in the above example, you cannot pass a null to it, otherwise, the result is unpredictable.

The following sample programs are written in VC 6.0. The functions of the dongle will be demonstrated step by step from an original program that is not encrypted yet. Developers who use other languages should also read this section carefully. There is no special development skill used below. The examples can be understood easily.

6.4.1 Unencrypted Program – Step 0

The following is a program that has not been encrypted:

```
#include <windows.h>
#include <stdio.h>

void main()
{
    // Anyone begin from here.
    printf("Hello FeiTian!\n");
}
```

6.4.2 Finding Dongle – Step 1

At the beginning of the program, a segment of an operation of finding a dongle with specified passwords is inserted. If the dongle is found, the program will proceed. Otherwise, the program will exit.

```
#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()
{
```

```
// =====
WORD retcode;
WORD handle, p1, p2, p3, p4;
DWORD lp1, lp2;
BYTE buffer[1024];

p1 = 0xc44c;
p2 = 0xc8f8;
p3 = 0;           // Program needn't Password3, Set to 0
p4 = 0;           // Program needn't Password4, Set to 0

// Try to find specified ROCKEY4 SMART
retcode = Rocky(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Not found
{

    printf("ROCKEY not found!\n");
    return;
}

// =====
printf("Hello FeiTian!\n");
}
```

This is a simple encryption method. Only a function of the ROCKEY4 SMART API is used.

For details on the function, see function “RY_FIND” in Section 6.2 “ROCKEY4 SMART API Services”.

Try the above program to observe what happens when the dongle is connected and what happens when it is not connected.

6.4.3 Opening Dongle – Step 2

At the beginning of the program, an operation of opening a dongle with specific passwords is added. If the dongle can be opened, the program proceeds. Otherwise, the program exits.

```
#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()
{
    // =====
    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;                // ROCKEY4 SMART Variable
    BYTE buffer[1024];             // ROCKEY4 SMART Variable
```

```
p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2
p3 = 0;         // Program needn't Password3, Set to 0
p4 = 0;         // Program needn't Password4, Set to 0

// Try to find specified Rockey
retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Not found
{
    printf("ROCKEY not found!\n");
    return;
}

retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error

{
    printf("Error Code: %d\n", retcode);
    return;
}

// =====

printf("Hello FeiTian!\n");

retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    printf("Error Code: %d\n", retcode);
    return;
}

}
```

6.4.4 User Memory – Step 3/Step 4

The dongle is initialized using the Editor (refer to Chapter 5) or in API mode. Write “Hello FeiTian!” to the low address area and then read the string. See below in Step 3, Step 4.

Example: Initialize the dongle and write “Hello FeiTian!” – Step 3

```
#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File
```

```

void main()
{
    // =====
    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;                // ROCKEY4 SMART Variable
    BYTE buffer[1024];              // ROCKEY4 SMART Variable

    p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
    p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2
    p3 = 0;         // Program needn't Password3, Set to 0
    p4 = 0;         // Program needn't Password4, Set to 0

    // Try to find Rockey

    retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Not found
    {
        printf("ROCKEY not found!\n");
        return;
    }

    retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }

    p1 = 0;    // Pos
    p2 = 14;   // Length

    strcpy((char*)buffer, "Hello FeiTian! ");
    retcode = Rockey(RY_WRITE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }
    printf("Write: %s\n", buffer);

    retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {

```

```
        printf("Error Code: %d\n", retcode);
        return;
    }
}
```

In the above example (Step 3) we have written “Hello FeiTian!” into the dongle. Below in the following example (Step 4) the string will be read and displayed from the dongle dynamically.

Example: Read content from the memory – Step 4

```
#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"          // Include ROCKEY4 SMART Header File

void main()
{
    // =====
    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;               // ROCKEY4 SMART Variable
    BYTE buffer[1024];            // ROCKEY4 SMART Variable

    p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
    p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2
    p3 = 0;         // Program needn't Password3, Set to 0
    p4 = 0;         // Program needn't Password4, Set to 0

    // Try to find specified Rockey
    retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Not found
    {
        printf("ROCKEY not found!\n");
        return;
    }

    retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }

    p1 = 0;    // Pos
    p2 = 14;   // Length

    buffer[14] = 0;
```

```

retcode = Rockey(RY_READ, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}

// =====
printf("%s\n", buffer);

retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    printf("Error Code: %d\n", retcode);
    return;
}
}

```

6.4.5 Generating a True Random Number with Dongle–Step 5

Generate a random number at the beginning of the program. Write this random number to a fixed address of the memory area of the dongle. Verify if the data at the address is equal to the random number while the program is running. If the program is running on another computer in this process, a different random number must have been written to the address. Therefore, the ability to prevent tracing is enhanced.

```

#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()
{
    // =====
    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;                // ROCKEY4 SMART Variable
    BYTE buffer[1024];             // ROCKEY4 SMART Variable

    p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
    p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2
    p3 = 0;         // Program needn't Password3, Set to 0
    p4 = 0;         // Program needn't Password4, Set to 0
}

```

```
// Try to find specified Rockey
retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Not found
{
    printf("ROCKEY not found!\n");
    return;
}

retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}

retcode = Rockey(RY_RANDOM, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}
printf("Random:%04X,%04X,%04X,%04X\n", p1,p2,p3,p4);

sprintf(buffer, "%04X", p1);
p1 = 0;    // Pos
p2 = 4;    // Length
p3 = 1;
retcode = Rockey(RY_WRITE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}
printf("Write: %s\n", buffer);

p1 = 0;    // Pos
p2 = 4;    // Length

buffer[4] = 0;
retcode = Rockey(RY_READ, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
```

```

if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}
printf("Read: %s\n", buffer);

if(buffer)

    printf("Hello FeiTian!\n");
else

    exit(0);

retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    printf("Error Code: %d\n", retcode);
    return;
}
}

```

6.4.6 Seed Code – Step 6/Step 7

The return code of the fixed seed is read using the Editor (refer to Chapter 5) or in API mode.

The seed may also be passed in from inner a program as required. The seed code calculation is performed inside the dongle and the algorithm is confidential. Then you may verify the return codes or use the return codes in an encryption routine. See below in Step 6, Step 7.

Example: Read the return code of the fixed seed (0x12345678) – Step 6

```

#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()
{
    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;                // ROCKEY4 SMART Variable
    BYTE buffer[1024];             // ROCKEY4 SMART Variable

```

```
p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2
p3 = 0;         // Program needn't Password3, Set to 0
p4 = 0;         // Program needn't Password4, Set to 0

// Try to find specified Rockey
retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Not found
{
    printf("ROCKEY not found!\n");

    return;
}

retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}

//seed Rockey
lp2 = 0x12345678;
retcode = Rockey(RY_SEED, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}

printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);

// Close Rockey
retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    printf("Error Code: %d\n", retcode);
    return;
}

    printf("\n");
}
```

Example: Verify the return code to determine if the program can be continued – Step 7

```

#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()
{
    WORD retcode;

    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;                // ROCKEY4 SMART Variable
    BYTE buffer[1024];             // ROCKEY4 SMART Variable

    p1 = 0xc44c;                   // ROCKEY4 SMART Demo Password1
    p2 = 0xc8f8;                   // ROCKEY4 SMART Demo Password2
    p3 = 0;                        // Program needn't Password3, Set to 0
    p4 = 0;                        // Program needn't Password4, Set to 0

    // Try to find specified Rockey
    retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Not found
    {
        printf("ROCKEY not found!\n");
        return;
    }

    retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }

    //seed Rockey
    lp2 = 0x12345678;
    retcode = Rockey(RY_SEED, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }

    if (p1==0xD03A && p2==0x94D6 && p3==0x96A9 && p4==0x7F54)

```

```
        printf("Hello FeiTian!\n");
    else
    {
        printf("Hello error!\n");

        return;
    }

    // Close Rockey
    retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

    if (retcode)
    {
        printf("Error Code: %d\n", retcode);
        return;
    }
}
```

6.4.7 User ID – Step 8/Step 9

The user ID is written using the Editor (refer to Chapter 5) or in API mode. This can be used in the aspects of software versions, product kinds, or other encryption processing. See Step 8 and 9.

Note: The advanced passwords (level 2 passwords) are required for performing the operation at Step 8.

Example: Write a user ID to an initialized dongle – Step 8

```
#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()
{
    // =====
    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;               // ROCKEY4 SMART Variable
    BYTE buffer[1024];            // ROCKEY4 SMART Variable

    p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
    p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2
    p3 = 0x0799;    // ROCKEY4 SMART Demo Password3
    p4 = 0xc43b;    // ROCKEY4 SMART Demo Password4
```

```

// Try to find specified Rockey
retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Not found
{

    printf("ROCKEY not found!\n");
    return;

}

retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}

lp1 = 0x88888888;
retcode = Rockey(RY_WRITE_USERID, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}

printf("Write User ID: %08X\n", lp1);

retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    printf("Error Code: %d\n", retcode);
    return;
}

}

```

Example: Verify the user ID. If it is not 0x88888888, output “Hello DEMO!” – Step 9

```

#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()

```

```
{
    // =====

    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;                // ROCKEY4 SMART Variable
    BYTE buffer[1024];             // ROCKEY4 SMART Variable

    p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
    p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2

    p3 = 0;         // Program needn't Password3, Set to 0
    p4 = 0;         // Program needn't Password4, Set to 0

    // Try to find specified Rockey
    retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Not found
    {
        printf("ROCKEY not found!\n");
        return;
    }

    retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }

    lp1 = 0;
    retcode = Rockey(RY_READ_USERID, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }
    if (lp1==0x88888888)
        printf("Hello FeiTian!\n");
    else
    {
        printf("Hello DEMO!\n");
        return;
    }
    retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
```

```

    {
        printf("Error Code: %d\n", retcode);
        return;
    }
}

```

6.4.8 Setting Module Characters – Step 10/Step 11/Step 12

Configure module character values and properties using the Editor (refer to Chapter 5) or in API mode. Check if the module is valid and can be decreased in a user program. Determine whether to activate the associated application module. The module character value may also be used by the program. Check if the module is allowed to be decreased to limit the number of uses of software. See below.

Note: The advanced passwords (level 2 passwords) are required for performing the operation at Step 10.

Example: Set a module character for an initialized dongle (e.g. set Module 0 to be valid and the property to be not decreasable) – Step 10

```

#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"          // Include ROCKEY4 SMART Header File

void main()
{
    // =====
    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;               // ROCKEY4 SMART Variable
    BYTE buffer[1024];            // ROCKEY4 SMART Variable

    p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
    p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2
    p3 = 0x0799;    // ROCKEY4 SMART Demo Password3
    p4 = 0xc43b;    // ROCKEY4 SMART Demo Password4

    // Try to find specified Rockey
    retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Not found
    {
        printf("ROCKEY not found!\n");
        return;
    }
}

```

```
    }

    retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }

    p1 = 0;
    p2 = 3;
    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        printf("Error Code: %d\n", retcode);
        return;
    }
    printf("Set Moudle 0: Pass = %04X Decrease not allowed\n", p2);

    retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        printf("Error Code: %d\n", retcode);
        return;
    }
}
```

Example: Output “Hello FeiTian!” in a user program if the module 0 is valid; otherwise, the program stops running or exits – Step 11

```
#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()
{
    // =====
    WORD retcode;
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;                // ROCKEY4 SMART Variable
```

```

BYTE buffer[1024];           // ROCKEY4 SMART Variable

p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1
p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2
p3 = 0;         // Program needn't Password3, Set to 0
p4 = 0;         // Program needn't Password4, Set to 0

// Try to find specified Rockey
retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Not found
{
    printf("ROCKEY not found!\n");
    return;
}

retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}

p1 = 0;
retcode = Rockey(RY_CHECK_MODULE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
if (retcode)
{
    printf("Error Code: %d\n", retcode);
    return;
}

if (p2)
    printf("Hello FeiTian!\n");
else
    return;

retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    printf("Error Code: %d\n", retcode);
    return;
}

```

```
}  
}  
}
```

Example: At Step 10, set p2=3 (allowed number of uses) and p3=1 (degression allowed). That is to say, the number of uses of the software is limited to 3 through Module 0 (p1=0) The usage limitation is achieved as shown in another example - Step 12:

```
#include <windows.h>  
#include <stdio.h>  
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File  
  
void main()  
{  
    // =====  
    WORD retcode;  
    WORD handle, p1, p2, p3, p4;    // ROCKEY4 SMART Variable  
    DWORD lp1, lp2;               // ROCKEY4 SMART Variable  
    BYTE buffer[1024];            // ROCKEY4 SMART Variable  
  
    p1 = 0xc44c;    // ROCKEY4 SMART Demo Password1  
    p2 = 0xc8f8;    // ROCKEY4 SMART Demo Password2  
    p3 = 0;         // Program needn't Password3, Set to 0  
    p4 = 0;         // Program needn't Password4, Set to 0  
  
    // Try to find specified Rockey  
    retcode = Rockey(RY_FIND, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);  
    if (retcode) // Not found  
    {  
        printf("ROCKEY not found!\n");  
        return;  
    }  
  
    retcode = Rockey(RY_OPEN, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);  
    if (retcode) // Error  
    {  
        printf("Error Code: %d\n", retcode);  
        return;  
    }  
    p1 = 0;  
    retcode = Rockey(RY_CHECK_MODULE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4,  
buffer);
```

```

    if (retcode)
    {
        printf("Error Code: %d\n", retcode);
        return;
    }

    if (p2!=1)
    {
        printf("Update Please!\n");
        return;
    }

    if(p3==1)
    {
        p1=0;
        retcode = Rockey(RY_DECREASE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if(retcode)
        {
            printf("Error Code: %d\n", retcode);
            return;
        }
    }

    // =====

    printf("Hello FeiTian!\n");

    retcode = Rockey(RY_CLOSE, &handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)

    {
        printf("Error Code: %d\n", retcode);
        return;
    }
}

```

6.4.9 Dongle Cascading – Step 13

We can plug in more than one ROCKEY4 SMART dongle with the same passwords, either with identical type or not. Since each dongle has a unique hardware ID, it is possible for the program to identify which dongle is opened, and then further distinguish the type of the opened dongle. An example is shown below in Step 13:

```

#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

```

```
void main()
{
    int i, rynum;
    WORD retcode;
    WORD handle[16], p1, p2, p3, p4;    // ROCKEY4 SMART Variable
    DWORD lp1, lp2;                    // ROCKEY4 SMART Variable
    BYTE buffer[1024];                 // ROCKEY4 SMART Variable

    p1 = 0xc44c; // ROCKEY4 SMART Demo Password1
    p2 = 0xc8f8; // ROCKEY4 SMART Demo Password2
    p3 = 0;      // Program needn't Password3, Set to 0
    p4 = 0;      // Program needn't Password4, Set to 0

    // Try to find all Rockey
    for (i=0; i<16; i++)
    {
        if (0 == i)
        {
            retcode = Rockey(RY_FIND, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
            if (retcode == ERR_NOMORE)
                break;
        }
        else
        {
            // Notice : lp1 = Last found hardID
            retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
                if (retcode == ERR_NOMORE)
                    break;
        }

        if (retcode) // Error
        {
            printf("Error Code: %d\n", retcode);

            return;
        }

        printf("Found Rockey: %08X  ", lp1);

        retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode) // Error
        {
            printf("Error Code: %d\n", retcode);
```

```

        return;
    }

}
printf("\n");

rynum = i;

// Do our work
for (i=0;i<rynum;i++)
{
    // Read Rockey user memory
    p1 = 0;    // Pos
    p2 = 12;   // Length

    buffer[12] = 0;
    retcode = Rockey(RY_READ, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {

        printf("Error Code: %d\n", retcode);
        return;
    }
    printf("%s\n", buffer);    // Output

    lp1=0;
    retcode = Rockey(RY_READ_USERID, &handle[i], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }
    printf("Read User ID: %08X\n", lp1);

    p1=0;
    retcode = Rockey(RY_CHECK_MODULE, &handle[i], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }
}

```

```
        printf("Check Moudle 0: ");
        if (p2)
            printf("Allow    ");
        else
            printf("No Allow  ");
        if (p3)

            printf("Allow Decrease\n");
        else
            printf("Not Allow Decrease\n");

    }

    // Close all opened Rockey
    for (i=0;i<rynum;i++)
    {
        retcode = Rockey(RY_CLOSE, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)

            {
                printf("Error Code: %d\n", retcode);
                return;
            }
    }
}
```

A maximum of 16 dongles can be attached to the same computer at the same time. The program can be designed to selectively access any dongles.

In the above program, we define a handle array to save the opened dongle handles, and prepare for the next operation on the specified dongle. We open the dongle at the same time as we find it, and we close all opened dongle handles before the program exits. Developers are better off operating in this manner. For a large program, it is OK to open/close the dongle just once at the beginning/end of the program. Frequent opening and closing operations will reduce the performance of the program. Don't worry, because our opening is in shared mode. Even if another program of yours needs to open the dongle at the same time, the program will not be blocked.

Note: Functions RY_OPEN and RY_CLOSE are called above. The dongle

must be opened first for most operations, except for RY_FIND and RY_FIND_NEXT. This is similar to the operation on the disk files. You should close the dongle immediately after finishing dongle related operations.

You can find the source code of the above programming examples on the CD or from the installation directory under “Samples”.

6.5 Advanced Application Examples

To facilitate the developers to master programming with the dongle as soon as possible, in the following sections we provide several programming examples for references, according to the flexible extension of the basic functions. Note that these examples are provided in order to illustrate part of the functions provided by ROCKEY4 SMART, and the comprehensive and flexible applications. How to make good use of the dongle is not fully represented in the examples, and requires the developers to combine with the specific environment. The following examples are only for references. After all, the public experiences have no encryption strength. (If you are familiar with the API calling already, you may skip to Chapter 7 ROCKEY4 SMART Hardware Algorithms.)

6.5.1 User Memory Area Applications

Example – Step 14: To use the ROCKEY4 SMART dongle to encrypt the string “Hello FeiTian!”, usually you can write the string into a certain location in the dongle’s user memory space all together at one time. However, the security level can be further enhanced if you write every subarea separately and then combine them.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}
```

```
void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];
    BYTE buf[1024];

    int i, j;

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0;
    p4 = 0;

    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);

    retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i = 1;
    while (retcode == 0)
    {
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode == ERR_NOMORE) break;
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
```

```

        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        i++;
        printf("Find Rock: %08X\n", lp1);
    }
    printf("\n");

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

        p1 = 0;
        p2 = 10;
        p3 = 1;
        strcpy((char*)buffer, "Hello ");
        retcode = Rockey(RY_WRITE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Write: Hello \n");

        p1 = 12;
        p2 = 12;
        p3 = 1;
        strcpy((char*)buffer, "FeiTian!");
        retcode = Rockey(RY_WRITE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Write: FeiTian!\n");

        p1 = 0;
        p2 = 10;

        memset(buffer, 0, 64);
    }

```

```
retcode = Rockey(RY_READ, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Read: %s\n", buffer);

p1 = 12;
p2 = 12;

memset(buf, 0, 64);
retcode = Rockey(RY_READ, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buf);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Read: %s\n", buf);

printf("\n");
printf("%s\n", strcat(buffer,buf));

retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

}
```

Example – Step 15: To encrypt and control execution of different modules of the application with the user memory area of the ROCKEY4 SMART dongle, you can write a serial number to a location of the user memory area and verify it during the execution of different modules.

```

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j;

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;
    p4 = 0xc43b;

    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);

    retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i = 1;
    while (retcode == 0)
    {
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,

```

```
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i++;
    printf("Find Rock: %08X\n", lp1);
}
printf("\n");

for (j=0;j<i;j++)
{
    p1 = 0;

    p2 = 12;
    p3 = 1;
    strcpy((char*)buffer, "a1b2c3d4e5f6");

    retcode = Rockey(RY_WRITE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write:a1b2c3d4e5f6\n");

    p1 = 0;
    p2 = 2;

    memset(buffer, 0, 64);
    retcode = Rockey(RY_READ, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
```

```

        {
            ShowERR(retcode);
            return;
        }
        printf("Read: %s\n", buffer);

        if (!strcmp(buffer, "a1"))
            printf("Run Module 1\n");
        else

            break;

        p1 = 2;
        p2 = 2;

        memset(buffer, 0, 64);
        retcode = Rockey(RY_READ, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Read: %s\n", buffer);

        if (!strcmp(buffer, "b2"))
            printf("Run Module 2\n");
        else
            break;

        retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        printf("\n");

    }

```

```
}
```

Example – Step 16: To encrypt and control the number of uses with the user memory area, you can write a number to the area and decrease it when the application is executed. This step can be combined with Step 12 for encryption.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;

    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j, num;

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0;
    p4 = 0;

    {
        retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
        }
    }
}
```

```

        return;
    }
    printf("Find Rock: %08X\n", lp1);

    retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);

        return;
    }

    i = 1;
    while (retcode == 0)
    {
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode == ERR_NOMORE) break;
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

        if (retcode)
        {
            ShowERR(retcode);

            return;
        }

        i++;
        printf("Find Rock: %08X\n", lp1);
    }
    printf("\n");

    for (j=0;j<i;j++)
    {
        p1 = 0;
        p2 = 2;
        p3 = 1;
        strcpy((char*)buffer, "03");
        retcode = Rockey(RY_WRITE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,

```

```
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write: 03\n");
    p1 = 0;
    p2 = 1;
    memset(buffer, 0, 64);
    retcode = Rockey(RY_READ, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Read: %s\n", buffer);

    num=atoi(buffer);

    if(num)
    {
        printf("Hello FeiTian!\n");
        num--;
    }
    else
    {
        return;
    }
    p1 = 0;
    p2 = 1;
    sprintf(buffer, "%ld", num);
    retcode = Rockey(RY_WRITE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write: %ld\n", num);

    retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
```

```

    }
    printf("\n");

    }
}
}

```

6.5.2 Seed Applications

Example – Step 17: You may use different seed codes for different software modules or in different places in the application. Then verify the seed codes in the application.

```

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;

    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j;

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0;
    p4 = 0;

    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);
}

```

```
retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

i = 1;
while (retcode == 0)
{
    retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i++;
    printf("Find Rock: %08X\n", lp1);
}
printf("\n");

for (j=0;j<i;j++)
{
    lp2 = 0x12345678;
    retcode = Rockey(RY_SEED, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);
}
```

```

        if(p1==0xD03A && p2==0x94D6 && p3==0x96A9 && p4==0x7F54)
            printf("Hello Fei!\n");
        else
            break;
        lp2 = 0x87654321;
        retcode = Rockey(RY_SEED, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);

        if(p1==0xB584 && p2==0xD64F && p3==0xC885 && p4==0x5BA0)
            printf("Hello Tian!\n");
        else
            break;

        lp2 = 0x18273645;
        retcode = Rockey(RY_SEED, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);

        if(p1==0x2F6D && p2==0x27F8 && p3==0xB3EE && p4==0xBE5A)
            printf("Hello OK!\n");
        else
            break;

        retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("\n");
    }
}

```

Example – Step 18: A string is encrypted with a key and then decrypted. The seed code is

used to generate the key.

The following illustrates the function of seed code. In your development, the following should be divided into two parts: the encryption part should be done before the dongle is delivered; only the decryption part should be presented in user program.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    char str[20] = "Hello FeiTian!";
    DWORD mykey = 12345678;
    int n, slen;

    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i,j;

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;
    p4 = 0xc43b;

    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);

    retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
```

```

        return;
    }

    i = 1;
    while (retcode == 0)
    {
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode == ERR_NOMORE) break;
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        i++;
        printf("Find Rock: %08X\n", lp1);
    }
    printf("\n");

    for (j=0;j<i;j++)
    {
        // Encrypt my data
        slen = strlen(str);
        lp2 = mykey;
        retcode = Rockey(RY_SEED, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode) // Error
        {
            printf("Error Code: %d\n", retcode);
            return;
        }

        for (n=0;n<slen;n++)
        {
            str[n] = str[n] + (char)p1 + (char)p2 + (char)p3 + (char)p4;
        }

        printf("Encrypted data is %s\n", str);
    }

```

```
// Decrypt my data
lp2 = mykey;
retcode = Rockey(RY_SEED, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode) // Error
{
    printf("Error Code: %d\n", retcode);
    return;
}

for (n=0;n<slen;n++)
{
    str[n] = str[n] - (char)p1 - (char)p2 - (char)p3 - (char)p4;
}
printf("Decrypted data is %s\n", str);

retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

printf("\n");
}
}
```

6.5.3 User ID Applications

Example – Step 19: Developers may write the current date and time as the UID when initializing the dongle. When the software is running, compare the current system date and time with the UID. Typically, the current system date and time should be later than the UID.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}
```

```

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];
    BYTE buf[1024];

    int i, j;

    SYSTEMTIME st;

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;
    p4 = 0xc43b;

    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);
    retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i = 1;
    while (retcode == 0)
    {
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode == ERR_NOMORE) break;
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
    }
}

```

```
        retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        i++;
        printf("Find Rock: %08X\n", lp1);
    }
    printf("\n");

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

        lp1 = 0x20021101;
        retcode = Rockey(RY_WRITE_USERID, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        printf("Write User ID: %08X\n", lp1);

        lp1 = 0;
        retcode = Rockey(RY_READ_USERID, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Read User ID: %08X\n", lp1);

        sprintf(buffer,"%08X",lp1);
        GetLocalTime(&st);
        printf("Date:%04d%02d%02d\n",st.wYear,st.wMonth,st.wDay);

        sprintf(buf,"%04d%02d%02d",st.wYear,st.wMonth,st.wDay);
        if(strcmp(buf,buffer)>=0)
```

```

        {
            printf("ok!\n");
        }
        else

            break;

retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

printf("\n");

}
}

```

6.5.4 Module Applications

Example – Step 20: Multiple module encryptions allow you to use different dongle modules to control if the corresponding application modules can execute or be enabled.

```

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;

    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j;

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;

```

```
p4 = 0xc43b;

{
retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Find Rock: %08X\n", lp1);

retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);

    return;
}

i = 1;
while (retcode == 0)
{
    retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)

    {
        ShowERR(retcode);
        return;
    }

    i++;
    printf("Find Rock: %08X\n", lp1);
}
printf("\n");

for (j=0;j<i;j++)
{
    p1 = 0;
```

```

        p2 = 0x2121;
        p3 = 0;
        retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Set Moudle 0: Pass = %04X Decrease no allow\n", p2);

        p1 = 0;

        retcode = Rockey(RY_CHECK_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);

        if (retcode)

        {
            ShowERR(retcode);
            return;
        }
        printf("Check Moudle 0: ");
        if (p2)
            printf("Run Modul 1!\n");
        else
            break;

        printf("\n");

        p1 = 8;
        p2 = 0xFFFF;

        p3 = 0;
        retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Set Moudle 8: Pass = %04X Decrease no allow\n", p2);

        p1 = 8;
        retcode = Rockey(RY_CHECK_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3,

```

```
&p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Check Moudle 8: ");
    if (p2)
        printf("Run Modul 2!");
    else
        break;

    retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    printf("\n");
}
}
```

Example – Step 21: This program illustrates how to perform multi-module encryption and check the status of the modules. Many applications are segmented into program modules that users may choose or purchase separately. For example, a user may purchase three modules of a four-module application. Developers can specify that the three modules are valid in the dongle. If the user wants one more module, developers need only to produce a new dongle in which the required module is activated and deliver it to the user, thanks for the cascading functionality of the Rockey4 SMART dongle, without returning the previously delivered dongle, updating it, and sending it back.

```
#include <windows.h>
#include <stdio.h>
#include "Ry4S.h"           // Include ROCKEY4 SMART Header File

void main()
{
    int i, j, rynum;
```

```

WORD retcode;
DWORD HID[16];

WORD handle[16], p1, p2, p3, p4;    // ROCKEY4 SMART Variable
DWORD lp1, lp2;                    // ROCKEY4 SMART Variable
BYTE buffer[1024];                 // ROCKEY4 SMART Variable

p1 = 0xc44c; // ROCKEY4 SMART Demo Password1

p2 = 0xc8f8; // ROCKEY4 SMART Demo Password2
p3 = 0;      // Program needn't Password3, Set to 0
p4 = 0;      // Program needn't Password4, Set to 0

// Try to find all Rockey
for (i=0;i<16;i++)
{
    if (0 == i)
    {
        retcode = Rockey(RY_FIND, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode == ERR_NOMORE) break;
    }
    else
    {
        // Notice : lp1 = Last found hardID
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode == ERR_NOMORE) break;
    }

    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);
        return;
    }

    printf("Found Rockey: %08X\n", lp1);
    HID[i] = lp1; // Save HardID
    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode) // Error
    {
        printf("Error Code: %d\n", retcode);

        return;
    }
}
printf("\n");

```

```
rynum = i;

// Do our work
for (i=0;i<rynum;i++)
{
    printf("Rockey %08X module status: ", HID[i]);
    for (j=0;j<16;j++)
    {
        p1 = j;          // Module No
        retcode = Rockey(RY_CHECK_MODULE, &handle[i], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
        if (retcode)      // Error
        {
            printf("Error Code: %d\n", retcode);
            return;
        }
        if (p2) printf("O");
        else printf("X");
    }
    printf("\n");
}

// Close all opened Rocky

for (i=0;i<rynum;i++)
{
    retcode = Rockey(RY_CLOSE, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        printf("Error Code: %d\n", retcode);
        return;
    }
}

}
```

The above program searches all dongles with the same passwords attached to the computer and displays the status of each module. “O” means that the module has been activated; and “X” means that the module has not been activated. You can write some module values with the Editor and check them as illustrated here. Note that if the content of the module is 0, the module is inactive; otherwise, the module is active.

If a module is active, the module can be used. Check the status of the module in your programs. If a module is active for any one of the dongles, the corresponding program module is available.

6.5.5 Dongles with Same UID for Different Software Products

If your company has developed multiple software products but only used the dongles with one user ID (identical user ID dongles) the following solution can be used to realize the corresponding relationship between dongle and software.

Example – Step 22: Use the memory area of the dongle to recognize the relationship between a set of dongles and a product. For example, the content of the user memory area is “Ver10”, the corresponding software product is Software A.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    WORD handleEnd;

    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j;

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;
    p4 = 0xc43b;

    {
        retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    }
}
```

```
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);

    retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)

    {

        ShowERR(retcode);
        return;
    }

    i = 1;
    while (retcode == 0)
    {
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode == ERR_NOMORE) break;
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)

        {

            ShowERR(retcode);
            return;
        }

        i++;
        printf("Find Rock: %08X\n", lp1);
    }
    printf("\n");

    for (j=0;j<i;j++)
    {
        /*p1 = 0;

        p2 = 5;
```

```

        p3 = 1;
        strcpy((char*)buffer, "Ver10");
        retcode = Rockey(RY_WRITE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Write:%s\n",buffer);

        */
        p1 = 0;
        p2 = 5;

        memset(buffer, 0, 64);
        retcode = Rockey(RY_READ, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Read: %s\n", buffer);

        if (!strcmp(buffer,"Ver10"))
        {
            handleEnd=handle[j];
            break;
        }
    }

    { //=====A=====
buffer);
        retcode = Rockey(RY_RANDOM, &handleEnd, &lp1, &lp2, &p1, &p2, &p3, &p4,
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Random: %04X\n", p1);

        lp2 = 0x12345678;
        retcode = Rockey(RY_SEED, &handleEnd, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
    
```

```
        {
            ShowERR(retcode);
            return;
        }
        printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);

        retcode = Rockey(RY_CLOSE, &handleEnd, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        printf("\n");
    }
}
```

Example – Step 23: Recognize the relationship between a set of dongles and a software product with the user ID of the dongle. For example, if the encrypted user ID is “11111111” (hex), the corresponding software is Software A.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    WORD handleEnd;
    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j;
```

```

p1 = 0xc44c;
p2 = 0xc8f8;
p3 = 0x0799;
p4 = 0xc43b;

{

retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Find Rock: %08X\n", lp1);

retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

i = 1;
while (retcode == 0)
{
    retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);

        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i++;
    printf("Find Rock: %08X\n", lp1);
}

```

```
printf("\n");

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

    /*lp1= 0x11111111;
    retcode = Rockey(RY_WRITE_USERID, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write User ID: %08X\n", lp1);
    */

    lp1 = 0;
    retcode = Rockey(RY_READ_USERID, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode)
    {

        ShowERR(retcode);
        return;
    }
    if(lp1==0x11111111)
    {
        handleEnd=handle[j];
        break;
    }

}

{
    //=====A=====
    p1 = 0;
    p2 = 12;
    p3 = 1;
    strcpy((char*)buffer, "Hello Feitian!");
    retcode = Rockey(RY_WRITE, &handleEnd, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write: %s\n",buffer);
```

```

        p1 = 0;
        p2 = 12;

        buffer[512]=0;
        retcode = Rockey(RY_READ, &handleEnd, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Read: %s\n",buffer);

        retcode = Rockey(RY_RANDOM, &handleEnd, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Random: %04X\n", p1);

        lp2 = 0x12345678;
        retcode = Rockey(RY_SEED, &handleEnd, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);

        retcode = Rockey(RY_CLOSE, &handleEnd, &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        printf("\n");
    }
}
}

```


Chapter 7 ROCKEY4 SMART

Hardware Algorithms

You can write some self-defined algorithms to the ROCKEY4 SMART dongle, use the dongle to do some necessary calculations, or use the calculation results in the programs. Since the algorithm area of the dongle cannot be read, even the manufacturer cannot get the information on what has been written by a developer. Thus we realize the software encryption protection to the maximum level.

The algorithm can be defined and written with the Editor. Alternatively, you can use the API interface RY_WRITE_ARITHMETIC to write the algorithm. Actually, it is easy to develop and add an algorithm to the ROCKEY4 SMART dongle. **We provide detailed introduction below for the developers to understand and master the techniques.**

7.1 Introduction to Self-defined Algorithms

7.1.1 Instruction Format

All instructions must be in the format: $\text{reg1} = \text{reg2} \text{ op } \text{reg3}/\text{value}$, where reg1, reg2 and reg3 are registers; value is a number; and op is an operator. For example: $A = A + B$.

The ROCKEY4 SMART supports the following operations:

+: Addition

-: Subtraction

<: Cyclic left shift

*: Multiplication

^: XOR

&: And

|: Or

?: Comparison

value is a decimal figure between 0 and 63.

Note:

? is used to compare two operands. For example: if $C = A ? B$, the result is as follows:

C	A?B	B?A
A<B	0	FFFF
A=B	FFFF	FFFF
A>B	FFFF	0

0 or FFFF will be written to C.

An example of algorithm written to the dongle:

$A = A + B$, $B = B + E$, $C = A * F$, $D = B + C$, $H = H \wedge H$

A, B, C ... are all registers in the dongle. They are 16-bit units. There are 8 such registers inside the dongle, with identifiers of A, B, C, D, E, F, G, and H respectively.

7.1.2 Internal Algorithms & Application Interface

After the developers have written an algorithm, how can a program call the algorithm and get the result? The dongle is provided with 3 internal algorithms, which can be used to call the self-defined algorithms in API mode: RY_CALCULATE1, RY_CALCULATE2, and RY_CALCULATE3.

Basically, the usage of the 3 algorithms has no difference (we will describe their difference later). All of them use p1, p2, p3, and p4 for passing in/out parameters.

In:

Internal register A of dongle = p1 of user program

Internal register B of dongle = p2 of user program

Internal register C of dongle = p3 of user program

Internal register D of dongle = p4 of user program

The register variables whose values may vary depending on the algorithm that is used:

Internal register E

Internal register F

Internal register G

Internal register H

Out:

p1 of user program = Internal register A of dongle

p2 of user program = Internal register B of dongle

p3 of user program = Internal register C of dongle

p4 of user program = Internal register D of dongle

Thus, the registers A, B, C, and D are used as user interface variables; and the registers E, F, G, and H are used as internal variables.

7.1.3 Difference Between Three Algorithms

According to the previous introduction, we know that for all of the three algorithms, p1, p2, p3 and p4 correspond to registers A, B, C and D respectively. In fact, the key difference between the algorithms resides in registers E, F, G and H.

When a developer's internal program (in the dongle) is called, registers A, B, C and D have already been set with data passed in by p1, p2, p3 and p4 before the internal program is executed. But the contents of registers E, F, G and H are neither 0 nor some random number. They are initialized according to the calculation algorithms that are used. See below:

Table 7-1

Internal variables of dongle	RY_CALCULATE1
A	P1
B	P2
C	P3
D	P4
E	High 16 bits of hardware ID
F	Low 16 bits of hardware ID
G	Content of module (according to the module number passed in through *lp2)
H	Random number

Table 7-2

Internal variables of dongle	RY_CALCULATE2
------------------------------	---------------

A	P1
B	P2
C	P3
D	P4
E	Return code 1 of seed (according to the seed in *lp2)
F	Return code 2 of seed (according to the seed in *lp2)
G	Return code 3 of seed (according to the seed in *lp2)
H	Return code 4 of seed (according to the seed in *lp2)

Table 7-3

Internal variables of dongle	RY_CALCULATE3
A	P1
B	P2
C	P3
D	P4
E	Value in module *lp2
F	Value in module (*lp2 + 1)
G	Value in module (*lp2 + 2)
H	Value in module (*lp2 + 3)

7.1.4 API Interfaces of User Program

Below, let us further describe the detailed function explanation and the additional descriptions of these three self-defined algorithms, combining with the ROCKEY4 SMART API introduced in the previous chapter.

Function	RY_CALCULATE1
For	Perform a dongle operation as specified
Input Parameters	function = RY_CALCULATE1 *handle = handle of the dongle *lp1 = starting point of calculation *lp2 = module number *p1 = input value 1 *p2 = input value 2 *p3 = input value 3 *p4 = input value 4
Return Values	If 0 is returned, the operation is successful. Other values indicate errors. If successful: *p1 = return value 1

	<p>*p2 = return value 2 *p3 = return value 3 *p4 = return value 4</p>
Additional Descriptions	<p>For example: Algorithm in the dongle is $A = B + C$, then The result of calling with this function is $*p1 = *p2 + *p3$. For example: Algorithm in the dongle is $A = A + G$, then If $*p1 = 0$ when passing in, when returning the content of $*p1$ equals to the value in the module specified by $*lp2$. Here you should see that although you cannot read the content of a module directly, you could determine the content using an algorithm. If possible, you had better check the content out with an algorithm.</p>

Function	RY_CALCULATE2
For	Perform a dongle operation as specified
Input Parameters	<p>function = RY_CALCULATE2 *handle = handle of the dongle *lp1 = starting point of calculation *lp2 = seed *p1 = input value 1 *p2 = input value 2 *p3 = input value 3 *p4 = input value 4</p>
Return Values	<p>If 0 is returned, the operation is successful. Other values indicate errors. If successful: *p1 = return value 1 *p2 = return value 2 *p3 = return value 3 *p4 = return value 4</p>
Additional Descriptions	<p>When calling an algorithm using this function by the dongle, the initial values of registers E, F, G, and H are the return values of the seed in $*lp2$. In other words, the dongle calls function RY_SEED with the value of $*lp2$, and places the return codes into registers E, F, G, and H for further user processing.</p>

Function	RY_CALCULATE3
For	Perform a dongle operation as specified
Input Parameters	<p>function = RY_CALCULATE3 *handle = handle of the dongle *lp1 = starting point of calculation *lp2 = starting module number *p1 = input value 1 *p2 = input value 2 *p3 = input value 3 *p4 = input value 4</p>
Return Values	<p>If 0 is returned, the operation is successful. Other values indicate errors. If successful:</p>

	<p>*p1 = return value 1 *p2 = return value 2 *p3 = return value 3 *p4 = return value 4</p>
Additional Descriptions	<p>When calling an algorithm using this function by the dongle, the initial values of registers E, F, G, and H are the contents of 4 successive modules starting from the module specified by *lp2. For example: If *lp2 = 0, the initial values of registers E, F, G, and H when calling this function are: E = content of Module 0 F = content of Module 1 G = content of Module 2 H = content of Module 3</p> <p>Note: When the address of the module is greater than 63, the address will automatically go back to 0. For example, if *lp2 = 63, the initial values of registers E, F, G, and H when calling this function are: E = content of Module 63 F = content of Module 0 G = content of Module 1 H = content of Module 2</p>

7.2 Writing Self-defined Algorithms

7.2.1 Writing Algorithm

You can write an algorithm using the Editor. Alternatively, you can develop a program to write the algorithm with RY_WRITE_ARITHMETIC API function.

Function	RY_WRITE_ARITHMETIC
For	Write self-defined algorithms of developers
Input Parameters	<p>function = RY_WRITE_ARITHMETIC *handle = handle of the dongle *p1 = starting point of calculation *buffer = a string of instructions of the algorithm</p>
Return Values	If 0 is returned, the operation is successful; other values indicate errors.

For example:

```
strcpy(buffer, "A=A+E, A=A+F, A=A+G, A=A+H");
```

```
p1 = 3;
```

```
retcode= Rockey(RY_WRITE_ARITHMETIC, handle, &lp1, &lp2, &p1, &p2, &p3,
```

&p4, buffer);

Obviously, the algorithm to be written is stored into the buffer, with instructions separated by commas. The dongle will automatically set the first instruction as the beginning of the algorithm, and the last instruction as the end of the algorithm. In this case:

Address 3 of algorithm area: $A=A+E$

Address 4 of algorithm area: $A=A+F$

Address 5 of algorithm area: $A=A+G$

Address 6 of algorithm area: $A=A+H$

Address 3 is the starting point of the algorithm in dongle program area. Address 6 is the ending point of the algorithm in dongle program area. After the instruction at address 6 is executed, the dongle will return to the user program. If you want to call the program in the dongle, you must start from the starting point. Otherwise, you will get the result of 4 random numbers.

7.2.2 Instruction Conventions

There are some restrictions on the instructions of an algorithm. The restrictions (or conventions) are described in the following examples:

$A = A + B$ *Valid*

$D = D \wedge D$ *Valid*

$A = B$ *Invalid, must be in algorithm format, e.g. $A = B / B$*

$A = 0$ *Invalid, must be in algorithm format, e.g. $A = A \wedge A$*

$C = 3 * B$ *Invalid, constants must be postpositioned, e.g. $C = B * 3$*

$D = 3 + 4$ *Invalid, only one constant is allowed in an instruction*

$A = A / B$ *Invalid, the division operator is not supported*

$H = E * 200$ *Invalid, the constant must be less than 64*

$A = A * 63$ *It depends. Constants are not allowed in the first or last instruction. If this instruction appears in the middle, it is valid, if it is the first or last instruction, it is invalid.*

7.3 Examples of Use of User-defined Algorithms

7.3.1 Basic Application Examples

1) Algorithm 1

First, write an algorithm. Note that an algorithm should only be written during development, and it should not appear in the end user program.

```
p1 = 0;
strcpy(buffer, "H=H^H, A=A*23, F=B*17, A=A+F, A=A+G, A=A<C, A=A^D, B=B^B,
C=C^C, D=D^D");
```

```
retcode = Rockey(RY_WRITE_ARITHMETIC, handle, &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
```

Then, call the algorithm in the program:

```
lp1 = 0;           // starting point of the algorithm in the dongle
lp2 = 7;           // specified module number
p1 = 5;            // initial value of A
p2 = 3;            // initial value of B
p3 = 1;            // initial value of C
p4 = 0xffff;       // initial value of D
```

```
retcode = Rockey(RY_CALCULATE1, handle, &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
```

The dongle carries the following initial values, and starts to execute the instructions from algorithm area 0:

```
A = 5 (p1)
B = 3 (p2)
C = 1 (p3)
D = 0xffff (p4)
E = High 16 bits of ID
F = Low 16 bits of ID
```

G = Content of Module 7 (lp2)

H = Random number

If the content of Module 7 is 0x2121, the result will be:

$$((5*23 + 3*17 + 0x2121) < 1) \wedge 0xffff = 0xbc71$$

Example of Algorithm 1 – Step 24:

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j;

    char cmd[] = "H=H^H, A=A*23, F=B*17, A=A+F, A=A+G, A=A<C, A=A^D, B=B^B,
C=C^C, D=D^D";
    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;
    p4 = 0xc43b;

    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);
}
```

```
retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

i = 1;
while (retcode == 0)
{
    retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)

    {

        ShowERR(retcode);
        return;

    }

    i++;

    printf("Find Rock: %08X\n", lp1);
}
printf("\n");

for (j=0;j<i;j++)
{
    /*
    p1 = 7;
    p2 = 0x2121;
    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
```

```

        ShowERR(retcode);
        return;
    }
    printf("Set Moudle 7: Pass = %04X Decrease no allow\n", p2);
    printf("\n");
    */

    p1 = 0;
    strcpy((char*)buffer, cmd);
    retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write Arithmetic 1\n");

    lp1 = 0;
    lp2 = 7;

    p1 = 5;
    p2 = 3;

    p3 = 1;
    p4 = 0xffff;
    retcode = Rockey(RY_CALCULATE1, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Calculate Input: p1=5, p2=3, p3=1, p4=0xffff\n");

    printf("\n");
    printf("Result = ((5*23 + 3*17 + 0x2121) < 1) ^ 0xffff = 0xBC71\n");
    printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);

    retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

```

```
        printf("\n");  
  
    }  
  
}
```

2) Algorithm 2

Example of Algorithm 2 – Step 25: Write algorithm ("A=A+B, A=A+C, A=A+D, A=A+E, A=A+F, A=A+G, A=A+H"); according to input parameters and internal variables, the result is:

0x7b17

```
#include <windows.h>  
#include <stdio.h>  
#include <conio.h>  
#include "Ry4S.h"  
  
void ShowERR(WORD retcode)  
{  
    if (retcode == 0) return;  
    printf("Error Code: %d\n", retcode);  
}  
  
void main()  
{  
    WORD handle[16], p1, p2, p3, p4, retcode;  
    DWORD lp1, lp2;  
    BYTE buffer[1024];  
  
    int i, j;  
  
    char cmd1[] = "A=A+B, A=A+C, A=A+D, A=A+E, A=A+F, A=A+G, A=A+H";  
  
    p1 = 0xc44c;  
    p2 = 0xc8f8;  
    p3 = 0x0799;  
    p4 = 0xc43b;  
  
    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
```

```

    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);

    retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i = 1;
    while (retcode == 0)
    {
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode == ERR_NOMORE) break;
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        i++;

        printf("Find Rock: %08X\n", lp1);
    }
    printf("\n");

    for (j=0; j<i; j++)
    {
        /*
        lp2 = 0x12345678;

```

```
retcode = Rockey(RY_SEED, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);
printf("\n");

*/

p1 = 10;

strcpy((char*)buffer, cmd1);
retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Write Arithmetic 2\n");

lp1 = 10;
lp2 = 0x12345678;
p1 = 1;
p2 = 2;
p3 = 3;
p4 = 4;
retcode = Rockey(RY_CALCULATE2, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Calculate Input: p1=1, p2=2, p3=3, p4=4\n");

printf("\n");
printf("Result =d03a + 94d6 + 96a9 + 7f54 + 1 + 2 + 3 + 4=0x7b17\n");
printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);

retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
```

```

    }

    printf("\n");

}

}

```

3) Algorithm 3

Example of Algorithm 3 – Step 26: Write algorithm ("A=A+B, A=A+C, A=A+D, A=A+E, A=A+F, A=A+G, A=A+H"); according to input parameters and internal variables, the result is: 0x14

```

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j;

    char cmd2[] = "A=A+B, A=A+C, A=A+D, A=A+E, A=A+F, A=A+G, A=A+H";

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;
    p4 = 0xc43b;

    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

```

```
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);

    retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

    if (retcode)

    {
        ShowERR(retcode);
        return;
    }

    i = 1;
    while (retcode == 0)
    {
        retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode == ERR_NOMORE) break;
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        i++;

        printf("Find Rock: %08X\n", lp1);
    }
    printf("\n");

    for (j=0;j<i;j++)
    {
        /*
        p1 = 0;
```

```

        p2 = 1;
        p3 = 0;
        retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);

            return;
        }
        printf("Set Moudle 0: Pass = %04X Decrease no allow\n", p2);

        p1 = 1;
        p2 = 2;
        p3 = 0;
        retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Set Moudle 1: Pass = %04X Decrease no allow\n", p2);

        p1 = 2;
        p2 = 3;
        p3 = 0;
        retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Set Moudle 2: Pass = %04X Decrease no allow\n", p2);

        p1 = 3;
        p2 = 4;
        p3 = 0;
        retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;

```

```
    }
    printf("Set Moudle 3: Pass = %04X Decrease no allow\n", p2);
    printf("\n");

    /*
    p1 = 17;
    strcpy((char*)buffer, cmd2);
    retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write Arithmetic 3\n");

    lp1 = 17;
    lp2 = 0;
    p1 = 1;
    p2 = 2;
    p3 = 3;
    p4 = 4;
    retcode = Rockey(RY_CALCULATE3, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Calculate Input: p1=1, p2=2, p3=3, p4=4\n");

    printf("\n");
    printf("Result = 1+2+3+4+1+2+3+4=0x14\n");
    printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);

    retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    printf("\n");
}
}
```

7.3.2 Integrated Algorithm Application Examples

1) Example 1

Example – Step 27: First, obtain the hardware ID of the dongle by finding the dongle; obtain the hardware ID again using the function of Algorithm 1; if the two hardware IDs are not equal, the program is illegal.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD findlp1, truelp1;

    DWORD lp1, lp2;
    BYTE buffer[1024];

    int i, j;

    char cmd[] = "A=E|E,B=F|F,C=G|G,D=H|H";

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;
    p4 = 0xc43b;

    retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Find Rock: %08X\n", lp1);
    findlp1=lp1;
```

```
retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

i = 1;
while (retcode == 0)
{
    retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i++;

    printf("Find Rock: %08X\n", lp1);
}
printf("\n");

for (j=0;j<i;j++)
{
    /*
    p1 = 7;
    p2 = 0x2121;
    p3 = 0;

    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
}
```

```

    }
    printf("Set Moudle 7: Pass = %04X Decrease no allow\n", p2);
    p1 = 0;
    strcpy((char*)buffer, cmd);
    retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write Arithmetic 1\n");
    */

    lp1 = 0;
    lp2 = 7;
    p1 = 1;
    p2 = 2;
    p3 = 3;
    p4 = 4;
    retcode = Rockey(RY_CALCULATE1, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Calculate Input: p1=1, p2=2, p3=3, p4=4\n");
    printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);

    printf("\n");
    printf("Moudle 7 : %x\n", p3);
    truelp1=MAKELONG(p2,p1);

    printf("truelp1 : %x\n",truelp1);
    if (findlp1==truelp1)

        printf("Hello FeiTian!\n");
    else
        break;

    retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);

```

```
        return;
    }

    printf("\n");
}
}
```

2) Example 2

Example – Step 28: Obtain the return code of a particular seed using the function of Algorithm 2, and compare it with the return code obtained through the same seed when the program runs for the first time; if they do not match, the program is illegal.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];

    WORD rc[4];

    int i, j;

    char cmd1[] = "A=E|E,B=F|F,C=G|G,D=H|H";

    p1 = 0xc44c;
    p2 = 0xc8f8;
    p3 = 0x0799;
    p4 = 0xc43b;
```

```

retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Find Rock: %08X\n", lp1);

retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

i = 1;
while (retcode == 0)
{
    retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);

    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    i++;

    printf("Find Rock: %08X\n", lp1);
}
printf("\n");
for (j=0;j<i;j++)
{

```

```
    lp2 = 0x12345678;
    retcode = Rockey(RY_SEED, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);

    rc[0] = p1;
    rc[1] = p2;
    rc[2] = p3;
    rc[3] = p4;

    //  :

    p1 = 0;
    strcpy((char*)buffer, cmd1);
    retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);

        return;
    }
    printf("Write Arithmetic 2\n");

    lp1 = 0;
    lp2 = 0x12345678;
    p1 = 1;
    p2 = 2;
    p3 = 3;
    p4 = 4;
    retcode = Rockey(RY_CALCULATE2, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Calculate Input: p1=1, p2=2, p3=3, p4=4\n");
    printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);
```

```

    printf("\n");
    if(rc[0]==p1 && rc[1]==p2 && rc[2]==p3 && rc[3]==p4)

        printf("Hello FeiTian!\n");
    else
        break;

    retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    printf("\n");
}
}

```

3) Example 3

Example – Step 29: You can get the values stored in the 16 modules using the function of Algorithm 3. Remember that the modules may not be read, even with the advanced passwords. You may write some important data to the modules or perform some other operations.

```

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;

    DWORD lp1, lp2;

```

```
BYTE buffer[1024];

int i, j;

char cmd2[] = "A=E|E,B=F|F,C=G|G,D=H|H";

p1 = 0xc44c;
p2 = 0xc8f8;
p3 = 0x0799;
p4 = 0xc43b;

retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Find Rock: %08X\n", lp1);

retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);

    return;
}

i = 1;
while (retcode == 0)
{
    retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
    {
```

```

        ShowERR(retcode);
        return;
    }

    i++;

    printf("Find Rock: %08X\n", lp1);
}
printf("\n");

for (j=0;j<i;j++)
{
    /*
    p1 = 0;
    p2 = 1;
    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);

    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Set Moudle 0: Pass = %04X Decrease no allow\n", p2);

    p1 = 1;
    p2 = 2;
    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);

    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Set Moudle 1: Pass = %04X Decrease no allow\n", p2);

    p1 = 2;
    p2 = 3;
    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);

    if (retcode)

```

```
    {
        ShowERR(retcode);
        return;
    }
    printf("Set Moudle 2: Pass = %04X Decrease no allow\n", p2);

    p1 = 3;
    p2 = 4;
    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    printf("Set Moudle 3: Pass = %04X Decrease no allow\n", p2);
    // :
*/

    p1 = 0;
    strcpy((char*)buffer, cmd2);
    retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Write Arithmetic 3\n");

    lp1 = 0;
    lp2 = 0;
    p1 = 0;
    p2 = 0;
    p3 = 0;
    p4 = 0;
    retcode = Rockey(RY_CALCULATE3, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
}
```

```

printf("Calculate Input: p1=0, p2=0, p3=0, p4=0\n");

printf("\n");
printf("Moudle 0: %x\n",p1);
printf("Moudle 1: %x\n",p2);
printf("Moudle 2: %x\n",p3);
printf("Moudle 3: %x\n",p4);

retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

printf("\n");
}
}

```

4) Example 4

Example – Step 30: You can use all of the 3 algorithms in a program. In the program below, 4 algorithm segments are written to the dongle, and the 3 algorithms are all used.

```

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Ry4S.h"

void ShowERR(WORD retcode)
{
    if (retcode == 0) return;
    printf("Error Code: %d\n", retcode);
}

void main()
{
    WORD handle[16], p1, p2, p3, p4, retcode;
    DWORD lp1, lp2;
    BYTE buffer[1024];

```

```
int i, j;
int t1,t2,t3;

char cmd[] = "H=H^H, A=A*23, F=B*17, A=A+F, A=A+G, A=A<C, A=A^D, B=B^B,
C=C^C, D=D^D";
char cmd1[] = "A=A+B, A=A+C, A=A+D, A=A+E, A=A+F, A=A+G, A=A+H";
char cmd2[] = "A=A+B, A=A+C, A=A+D, A=A+E, A=A+F, A=A+G, A=A+H";
char cmd3[] = "H=H^H,A=A|A, B=B|B, C=C|C,D=A+B,D=D+C";

p1 = 0xc44c;
p2 = 0xc8f8;

p3 = 0x0799;
p4 = 0xc43b;

retcode = Rockey(RY_FIND, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Find Rock: %08X\n", lp1);

retcode = Rockey(RY_OPEN, &handle[0], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}

i = 1;
while (retcode == 0)
{
    retcode = Rockey(RY_FIND_NEXT, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode == ERR_NOMORE) break;
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }

    retcode = Rockey(RY_OPEN, &handle[i], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
    if (retcode)
```

```

        {
            ShowERR(retcode);
            return;
        }

        i++;

        printf("Find Rock: %08X\n", lp1);
    }

    printf("\n");

    for (j=0;j<i;j++)
    {
        p1 = 7;
        p2 = 0x2121;
        p3 = 0;
        retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Set Moudle 7: Pass = %04X Decrease no allow\n", p2);
        printf("\n");

        lp2 = 0x12345678;
        retcode = Rockey(RY_SEED, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Seed: %04X %04X %04X %04X\n", p1, p2, p3, p4);
        printf("\n");

        p1 = 0;
        p2 = 1;
        p3 = 0;
        retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {

```

```
        ShowERR(retcode);
        return;
    }
    printf("Set Moudle 0: Pass = %04X Decrease no allow\n", p2);

    p1 = 1;
    p2 = 2;

    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Set Moudle 1: Pass = %04X Decrease no allow\n", p2);

    p1 = 2;
    p2 = 3;
    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Set Moudle 2: Pass = %04X Decrease no allow\n", p2);

    p1 = 3;
    p2 = 4;
    p3 = 0;
    retcode = Rockey(RY_SET_MODULE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
    if (retcode)
    {
        ShowERR(retcode);
        return;
    }
    printf("Set Moudle 3: Pass = %04X Decrease no allow\n", p2);
    printf("\n");

    p1 = 0;
    strcpy((char*)buffer, cmd);
```

```

        retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Write Arithmetic 1\n");

        lp1 = 0;
        lp2 = 7;
        p1 = 5;
        p2 = 3;
        p3 = 1;
        p4 = 0xffff;
        retcode = Rockey(RY_CALCULATE1, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Calculate Input: p1=5, p2=3, p3=1, p4=0xffff\n");

        printf("Result = ((5*23 + 3*17 + 0x2121) < 1) ^ 0xffff = 0xBC71\n");
        printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);
        t1=p1;

        p1 = 10;
        strcpy((char*)buffer, cmd1);
        retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Write Arithmetic 2\n");

        lp1 = 10;
        lp2 = 0x12345678;
        p1 = 1;

```

```
p2 = 2;

p3 = 3;
p4 = 4;
retcode = Rockey(RY_CALCULATE2, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Calculate Input: p1=1, p2=2, p3=3, p4=4\n");

printf("Result =d03a + 94d6 + 96a9 + 7f54 + 1 + 2 + 3 + 4=0x7b17\n");
printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);
t2=p1;

p1 = 17;
strcpy((char*)buffer, cmd2);
retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Write Arithmetic 3\n");

lp1 = 17;
lp2 = 0;
p1 = 1;
p2 = 2;
p3 = 3;
p4 = 4;
retcode = Rockey(RY_CALCULATE3, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
if (retcode)
{
    ShowERR(retcode);
    return;
}
printf("Calculate Input: p1=1, p2=2, p3=3, p4=4\n");
printf("Result = 1+2+3+4+1+2+3+4=0x14\n");
printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);
```

```

        t3=p1;

        printf("\n");
        p1 = 24;
        strcpy((char*)buffer, cmd3);
        retcode = Rockey(RY_WRITE_ARITHMETIC, &handle[j], &lp1, &lp2, &p1, &p2, &p3,
&p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }
        printf("Write Arithmetic \n");

        lp1 = 24;
        lp2 = 7;
        p1 = t1;
        p2 = t2;
        p3 = t3;
        p4 = 0;
        retcode = Rockey(RY_CALCULATE1, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4,
buffer);
        if (retcode)
        {
            ShowERR(retcode);
            return;
        }

        printf("Calculate Output: p1=%x, p2=%x, p3=%x, p4=%x\n", p1, p2, p3, p4);

        retcode = Rockey(RY_CLOSE, &handle[j], &lp1, &lp2, &p1, &p2, &p3, &p4, buffer);
        if (retcode)
        {
            ShowERR(retcode);

            return;
        }
        printf("\n");
    }
}

```

7.3.3 Advanced Algorithm Application Examples

Example – Step 31: The key calculations in the source code are completely performed by the

dongle. The following example illustrates this by 3 stages: Firstly, the source code before encryption is shown; secondly, the dongle is initialized; at last, it is the program used by the end users after encryption.

Source program:

```
#include "stdafx.h"
#include "DrawCircle.h"

#include "DrawCircleDoc.h"
#include "DrawCircleView.h"
#include "DrawParamDlg.h"
#include "DrawMethodDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

void CDrawCircleView::DrawCircleMidPoint(CDC *pDC, int iCenterX, int iCenterY, int r)
{
    int x=0;
    int y=r;
    int p=1-r;

    TRACE("Origin\n");

    CirclePlotPoints(pDC,iCenterX,iCenterY,x,y);

    m_lpCircleBuf[0].x = x;
    m_lpCircleBuf[0].y = y;
    m_nPointCount=1;

    while(x<y)
    {
        x++;
        if(p<0)
        {
            p+=2*x+1;
        }
        else
        {
            y--;
            p+=2*(x-y)+1;
        }
    }
}
```

```

        TRACE("%d,(%d,%d);",p,x,y);
        CirclePlotPoints(pDC,iCenterX,iCenterY,x,y);

        m_lpCircleBuf[m_nPointCount].x = x;
        m_lpCircleBuf[m_nPointCount].y = y;
        m_nPointCount++;
    }
    TRACE("\n");
}

```

Initialize the dongle:

```

#include "stdafx.h"
#include <windows.h>
#include "..\inc\Ry4S.h"

void ReportErr(WORD wCode)
{
    printf("ERROR:%d\n",wCode);
}

int main(int argc, char* argv[])
{
    WORD  p1=0xc44c,p2=0xc8f8,p3=0x0799,p4=0xc43b;
    DWORD lp1,lp2;
    WORD  handle[16];
    BYTE  buffer[1024];
    BYTE  cmdstr[] = "B=B|B,B=B+1,B=B*2,B=B+1,A=A+B,C=C-1,C=C*2,B=A-C";
    WORD  retcode;

    retcode = Rockey(RY_FIND,&handle[0],&lp1,&lp2,&p1,&p2,&p3,&p4,buffer);
    if(retcode)
    {
        ReportErr(retcode);
        return 0;
    }

    printf("Find successfully\n");

    retcode = Rockey(RY_OPEN,&handle[0],&lp1,&lp2,&p1,&p2,&p3,&p4,buffer);
    if(retcode)
    {
        ReportErr(retcode);
        return 0;
    }
}

```

```
printf("Open successfully\n");

p1 = 10;
retcode =
Rockey(RY_WRITE_ARITHMETIC,&handle[0],&lp1,&lp2,&p1,&p2,&p3,&p4,cmdstr);
if(retcode)
{
    ReportErr(retcode);
    return 0;
}
printf("Write arithmetirc successfully\n");

retcode = Rocky(RY_CLOSE,&handle[0],&lp1,&lp2,&p1,&p2,&p3,&p4,buffer);

return 0;
}
```

End user program:

```
#include "stdafx.h"
#include "DrawCircle.h"

#include "DrawCircleDoc.h"
#include "DrawCircleView.h"
#include "DrawParamDlg.h"
#include "DrawMethodDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

WORD    p1=0xc44c,p2=0xc8f8,p3=0x0799,p4=0xc43b;
DWORD  lp1,lp2;
WORD   handle[16];
BYTE   buffer[1024];

void CDrawCircleView::DrawCircleMidPoint_Rockey(CDC *pDC, int iCenterX, int iCenterY, int r)
{
    int x=0;
    int y=r;
    int p=1-r;
    int seed=0;

    short p1,p2,p3,p4;
```

```

CirclePlotPoints(pDC,iCenterX,iCenterY,x,y);

TRACE("Hardware\n");
m_lpCircleBuf[0].x = x;
m_lpCircleBuf[0].y = y;
m_nPointCount=1;

while(x<y)
{
    p1 = p;
    p2 = x;
    p3 = y;
    p4 = seed;

    if(!RunRockey((WORD&)p1,(WORD&)p2,(WORD&)p3,(WORD&)p4))
    {
        // AfxMessageBox("Runtime error");
        break;
    }

    if(p<0)
    {
        p = p1;
    }
    else
    {
        p = p2;
        y--;
    }

    x++;
    TRACE("%d,(%d,%d);",p,x,y);
    CirclePlotPoints(pDC,iCenterX,iCenterY,x,y);

    m_lpCircleBuf[m_nPointCount].x = x;
    m_lpCircleBuf[m_nPointCount].y = y;
    m_nPointCount++;
}
TRACE("\n");
}

BOOL CDrawCircleView::RunRockey(WORD &A, WORD &B, WORD &C, WORD &D)
{
    WORD retcode;

    lp1 = 10;
    retcode = Rockey(RY_CALCULATE1,&handle[0],&lp1,&lp2,&A,&B,&C,&D,buffer);

```

```
    if(retcode)
        return FALSE;
    else
        return TRUE;
}
```

7.4 Considerations

The algorithm area in the ROCKEY4 SMART has 128 words. That is to say, the ROCKEY4 SMART dongle supports as many as 128 instructions or 128 algorithms, because every instruction takes one word. Developers do not need to consider the starting and ending flags of an algorithm, because they are handled by the dongle. In practice, this means that if you write a 2-instruction algorithm to the dongle, and then a 3-instruction algorithm, the two algorithms will not be recognized as a single 5-instruction algorithm. If the starting point used in calculations is incorrect, the result will be unpredictable.

7.5 Application Tips of Encryption Solution

1. **Use as more as possible encryption calls to the ROCKEY4 SMART API** - In the program need to be encrypted, insert multiple calls to the ROCKEY4 SMART API from within your application to increase the cracker's work strength, and the complex multiple calls to the API can increase the cracking difficulties. The more calling to APIs and verifying the return codes, the more difficulties to crack these codes. Of course, these calling should reside in as many different places as possible in your program, which needs to be encrypted.
2. **Dynamically use the seed code functions as far as possible** - In the program need to be encrypted, dynamically call the seed code functions in the dongle, can make the information to be verified or the information exchanged with the dongle every time inconsistent with each other. This is even more difficult to the crackers. For example,

the developers can randomly obtain some data of the system as the seed code, like system date, etc.

3. **Avoid using the duplicate encryption methods in your application as far as possible** - If you repeatedly use the same protection method in your application, it will be easier for the cracker to find the rule and crack your application. In other words, if a developer uses different encryption methods in one program, every time the cracker will spend more energy to trace the program, and constantly face new cracking tasks. If the developer uses some dynamic encryption methods, the cracking will be even more difficult.
4. **Encrypt some character string and data** – Encrypt some character strings and data in the program need to be encrypted, to let the decryption rely on the existence of the ROCKEY4 SMART dongle. If these character strings and data cannot be properly decrypted, program failure or illegality will be caused. (Please refer to the example in Step 18 for specific applications)
5. **Use API encryption and Envelope encryption together** – The best protection will be achieved by using a complex and dynamic implementation of the ROCKEY4 SMART API, and then protecting the new executable files with the envelope.

In summary, the encryption strength largely depends on the practical environment of how the encryption methods are used. It is recommended that the encryption developers using the dongle flexibly apply our encryption tools according to their practical environment.

Chapter 8 FAQs

This chapter covers some frequently asked questions and some solutions when using the ROCKEY4 SMART dongles.

8.1 Common Ways of Dealing with Problems

- Test the dongle with Ry4S_Editor.exe under directory TOOLS
- Use the latest version of SDK (Download from our website)
- Please refer to our website at <http://www.ftsafe.com>, we will update this website frequently.
- Use a replacement computer to test if the problem still exists.
- Check if the computer has been infected with viruses, which may prevent the program from running normally.

8.2 FAQs

1. What is the evaluation kit?

Answer: The evaluation kit includes a dongle provided to developers for evaluation purpose. It also includes a product package, a manual, and a CD etc. The dongle is the same as the formal version, except that its passwords are public and unified.

2. Is the password mechanism of the ROCKEY4 SMART dongle secure?

Answer: It is very secure. The users can use our tool to generate 4 passwords in two levels, each has a length of 16 bits. The first level is the basic passwords, which are used to perform basic operations on the dongle. The second level is the advanced passwords, which are specially used by the developers for controlling writing to the dongle and defining encryption algorithms. The advanced passwords must not appear in the software delivered to end users, so that they cannot be obtained even by tracing. Moreover, if the advanced passwords are incorrect and the operation of writing to the high address

area of the user memory is attempted for 4 times in succession, the dongle will be locked for 2 seconds. During the locking period, no operation will be accepted. This measure prevents hackers from using a program for password attempts.

3. What is the dongle with the same passwords?

Answer: After a batch of dongles is delivered to the developers, the users may modify the passwords of the dongles to be identical, using our password initialization tool. After the encryption task is done, the developers produce their own software in a large amount by compressing it into CDs. Then the developers sell the encrypted software out together with the dongles, eliminating the needs to recompile every set of software one by one.

4. Is it true that a data sharer can share a dongle?

Answer: The sharer can be prevented by using our methods. Actually, it is very simple. You can design your program to generate a random number and write it to a fixed address on the dongle when the program starts, and verify the content at that address with the random number during the runtime. If during this time the program is also running on another computer, a different random number must have been written to the fixed address.

5. Will the performance of the software be compacted significantly if I write a complex algorithm to the ROCKEY4 SMART dongle?

Answer: No. During the testing, we got that the running time difference between the simplest algorithm and the most complex algorithm is merely tens of milliseconds. The difference will not be perceived at all if the algorithm is not called frequently.

6. Why is my USB dongle recognized as an unknown device?

Answer: This problem occurs rarely. Interference may exist in your environment or your dongle is not in good connection with your computer. Please remove and attach your dongle again.

7. My computer is equipped with a USB port and installed Windows 98. Why can't I see the USB device in Device Manager?

Answer: Maybe the USB support option is disabled in BIOS settings.

8. How can I update the software of the dongle?

Answer: If you are our evaluation user, you will receive updates from us regularly. Otherwise, go to

<http://www.ftsafe.com> to download the latest version of developer's kit.

9. Why did I fail to call RY_ADJUST_TIMER?

Answer: A time parameter of SYSTEMTIME type is required to call this function. There is a time stamp in the dongle. If the parameter is earlier (or less) than the time stamp, modification is not possible. A failure will be returned in this case. If the value of the parameter is later (or greater) than the time stamp, the value of the parameter will be assigned to the time stamp, and calling to the function will succeed. This function is often used to check if the computer time has been changed.

Appendix A Directory Structure Of CD

On the ROCKEY4 SMART CD, you can find a complete ROCKEY4 SMART developer's kit, which is briefly introduced here in the following table.

Directory	Description
API	Static libraries, dynamic libraries, COM components, OCX controls required for development
Docs	Documentation that describes the calling interface of the dongle functions and the usage of tools
Driver For Win98	HID driver for Windows 98
Include	Header files required for development
Management	License manager software, production tool software
Samples	Examples of various programming languages
Support	Jet driver with an Access database for Windows 98
Tools	Some software tools, including the Editor, the flash encryption tool, the envelope encryption tool and the .Net program encryption tool