FEITIAN
WE BUILD SECURITY

ROCKEY5 TIME USER GUIDE

V1.0

Feitian Technologies Co. Ltd

Website：http://www.ftsafe.com/

Histroy

Date	Version	Content
Nov 2011	V1.0	The first version

Software Developer's Agreement

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

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

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

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

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

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

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

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

advised of the possibility of damages, or for any claim by you based on any third-party claim.

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

Content

Quick Start and Hints

■ ROCKEY5 TIME factory default:
Product vendor ID: "00000000".
Developer password: "123456781234567812345678".
No maximum retries limits.
If you have set retry limit of developer password in the root, the developer password will be locked when the user exceeds the limits when continuous unsuccessful verification. In that case, the product is required to be recalled back to the manufacturer.

■ ROCKEY5 TIME is HID device with USB interface, supporting Windows operating system such as Win2000,XP, Vista and Win 7. And none of them requires extra driver to be installed.

■ ROCKEY5 TIME has the unique global hardware ID (64 bit/8 bytes).

■ ROCKEY5 TIME user space is 32k. Rewritten time reaches 100 thousand, no limit on reading. Working temperature: 0℃－70℃.

■ In "Utilities" folder of SDK, you can find the ROCKEY5 TIME user tool "RY5Tools.exe", which can offer multiple functions like initialization, editor and modification .etc.

■ In "Utilties" folder of SDK, you can find the ROCKEY5 TIME virtual file system manager (VfsSet.exe). It simulates as a real dongle environment and one virtual file representing one ROCKEY5 TIME dongle.

■ In "Utilties\Envelope" folder of SDK, you can find the enveloper encryption tool (Envelop.exe), as a pure simple and quick way to encrypt the Win32 and .Net application without any other assistance.

■ ROCKEY5 TIME also provides API for functions to encrypt, you can integrate the APIs into the application intended to be protected. In this way, all the functionality of ROCKEY5 TIME can be developed to provide an extremely high level security.

■ ROCKEY5 TIME provides abundant programming interfaces for almost all kinds of developer tool. The relevant library files can be found in "API" folder of SDK.

API\BCB_Delphi\RY5_cbc.lib   C++ Builder Static Link Library

API\Com\ Rockey5Com.dll   Com components, pre-register is required:

1. Copy Rockey5Com.dll to (%Windows%\system32\)

2. Register in terminal: regsvr32 Rockey5Com.dll

3. Unstall component: regsvr32 /u Rockey5Com.dll

API\BCB_Delphi\ROCKEY5_Delphi.obj Delphi Library file

API\Dynamic\ Rockey5.dll Dynamic link library

API\Dynamic\ Rockey5.lib LIB file for dynamic link library

Include\ RY5_API.h Header file

API\Java\ RY5Java.dll   JAVA' s jni interface

API\Java\ RY5jni.jar   JAVA library file

API\Static\Rockey5.lib C static link library.

■ "C51" folder in SDK stores C51 program's library and header file

■ If you need ROCKEY5 TIME to develop or test, please contact the company from website: http://www.FTsafe.com

Chapter 1. ROCKEY5 TIME Introduction

1.1 About ROCKEY5 TIME

The ROCKEY5 TIME is based on smart card technology, enduring 100 thousand hardware writting, over 10 years preserve, working temperature: 0℃－70℃.

ROCKEY5 TIME integrates dongle and smart card technology, with driverless design, multiple functions and cost effective pricing, targeting various areas such as software proctection, authentication, E-business and information security.

1.2 ROCKEY5 TIME Features

■ **Compact Design**

The dongle is compact and portable.

■ **High Speed**

ROCKEY5 TIME was designed to process even very complex algorithms with minimal delay for your application. Users will typically notice no degradation in application performance as a result of ROCKY5 TIME being implemented.

■ **Ease of Use**

ROCKEY5 TIME reduced API set simplifies the programming effort. User utilities are also provided in order to make the use of ROCKEY TIME more convenient.

■ **High Security Levels**

ROCKEY5 TIME is a completely new model which offers a much higher level of security. ROCKEY5 TIME implements a two level security system to segregate users who require read-only access from those who require administrative privileges.
ROCKEY5 TIME adopts scrambling technology to communication. It can prevent the dongle from being shared by multiple computers and the USB communication cannot be simulated. It supports multi-threads access.
ROCKEY5 TIME generates Vendor ID by seed codes. The maximum length of seed is 250 bytes. This enhances the security greatly.
ROCKEY5 TIME has a built-in smart card chip and C51 virtual machine. Kernel algorithms and data can be put and executed inside the dongle. Thus ROCKEY5 TIME works as a small computer system and exchange data with PC over USB interface. If the algorithm is enough complex, it’s impossible to

■ **Internal Hardware Timer**

Time chip is embedded in ROCKEY5 TIME, supporting the expiration mode and remainder time mode. When the software is expired, the user will be unable to use the software except remote updating from developer to restart the function.

■ **Comprehensive System Support**

The ROCKEY5 TIME dongle supports various Windows operating systems. The encrypted application programs support the following platforms: Windows 2000/XP/Server 2003/Vista/2008/Windows 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, C51, etc.

1.3 Software Protection

ROCKEY TIME uses many advanced encryption technologies and it's the top software protection dongle.

■ **Hardware security**

ROCKEY5 TIME dongle uses smart card as the hardware kernel. Smart card is used in bank, finance and other fields. Security is the most important. Smart card factory uses special hardware design to ensure anti-crack, anti-trace and other safety precautions. The new generation smart card contains CPU, RAM, EPROM, FLASH and other modules. It's same with a mini computer. This feature is the foundation for us to use vary complicate security protocol.

■ **Hardware compatibility**

ROCKEY5 TIME dongle is designed as driverless HID device without any requirement for additional driver installation but provides high speed for application.

■ **Software security**

The user's kernel algorithm can be put in the smart card and runs inside the smart card. Under the smart card OS, the program can be partly executed in ROCKEY5 TIME and the algorithm only exchange data with computer through USB port. The developer's program is unable to be cracked due to its invisibility in computer and impossible to get the code running in ROCKEYe3 TIME.

Chapter 2. Basic Concepts of ROCKEY5 TIME

2.1 Developer Password

ROCKEY5 TIME provides Developer Password for software manufacturers. When a ROCKEY5 TIME is re-attached to a PC, the security level will be reset to no authenticated status.

Please note: When a ROCKEY5 TIME dongle is closed, such as calling RY5_Close(RY_HANDLE handle, BOOL IsReset) function, if the IsReset parameter is TRUE, the data stored in device memory and the security status will be all cleaned. If the IsReset is FALSE, nothing will be cleaned.

Developer Password (24 bytes) is used for software manufacturers. The main function of Developer Password is to configure the ROCKEY5 TIME hardware, such as creating files, deleting files and so on. The length of the Developer Password is 24 bytes. The default value is "123456781234567812345678" and the hexadecimal value is "0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38". With the Developer Password, customers can create and delete files but cannot get any internal data from dongle. So it is strongly recommended to change the Developer Password before using in order to prevent being obtained illegally.

Notes on Developer Password:

■ The length of Developer Password must be 24 bytes.

■ There is no retry limitation for Developer Password. To avoid brute-force attack, we recommend customers to set a maximum retry times. This value can be set from 1 to 254 (Please refer chapter 7.1.10, the detailed information for RY5_ChangePin). If customers set this value to 255(0xFF), it means that there is no retry limitation. Please note: If the maximum retry time has been reached, the ROCKEY5 TIME will be locked. In this case, the only way to unlock the dongle is to send back to FEITIAN. So please keep the Developer Password carefully.

■ In case of losing the Developer Password, the ROCKEY5 TIME can also keep the protected software secure. With the Developer Password, illegal users can only write or delete data in ROCKEY5 TIME but cannot get any data from ROCKEY5 TIME. So they cannot use simulator to attack ROCKEY5 TIME. But we still suggest developers keep their Developer Password carefully.

2.2 Vendor ID (VID)

The default value of Vendor ID (8 bytes long) is "00000000". Software manufacturers can use ROCKEY5 TIME utilities or API to set the Vendor ID. The Vendor ID is used for software manufacturers to check whether the current ROCKEY5 TIME belongs to them. To avoid others setting same Vendor ID, ROCKEY5 TIME uses seed code algorithm to calculate the Vendor ID. ROCKEY5 TIME takes an input code (maximum length is 250 bytes) in and work out a Vendor ID. This process cannot be reversed. Even if an illegal user got the Vendor ID, he cannot copy a same ROCKEY5 TIME due to he desn't know the seed code. This enhances the security level greatly.

Software manufacturers can also use RY5_SetVendorID interface to set Vendor ID. Please refer to section 7.1.9.

2.3 Hardware ID (HID)

Each ROCKEY5 TIME has a global unique hardware ID. This ID cannot be modified. When software manufacturers want to protect software for a particular customer, they can use this HID to achieve this goal.

Software manufactures can use RY5_GetHardID interface to get Hardware ID. Please refer to section 7.1.4.

2.4 Data Memory

ROCKEY5 TIME provides 8 KB data memory space. Bytes from 0 to 7168 belong to lower data memory zone. Bytes from 7169 to 8192 belong to upper data memory zone. Both software manufacturers and end users can read/write in lower data memory zone. While upper data memory zone only allows software manufactures to write after verifying Developer Password . But both the 2 roles are allowed to read data from upper data memory zone even without verifying Developer Password.

Software manufacturers can use RY5_VendorWrite interface to write date to data memory. Please refer to section 7.1.13.

2.5 Shared Memory

ROCKEY5 TIME contains a 32KB Shared Memory zone which enables all internal programs to run with sufficient memory.

Software manufacturers can use RY5_ReadShare and RY5_WriteShare to read and write shared memory zone. Please refer to section 7.1.4.

2.6 Hardware Based Clock

ROCKEY5 TIME contains a hardware based clock chip and a battery. The UTC time has been set when produced. Both end users and software manufacturers cannot modify it. The clock chip cannot be duplicated or simulated due to its security design. Software manufacturers can use RY5_GetRTCTime interface or C51 time interface to get internal UTC time. Please refer section 7.1.32 or 6.9.1.

2.7 User Memory

ROCKEY5 TIME provides 32KB user memory. Users can store their files here. Please refer section 2.7.

2.8 ROCKEY5 TIME File System (R5FS)

ROCKEY5 TIME file system is similar to Windows file system. The R5FS includes executable files, data files and key files. Each file has a unique File ID. The File ID is 2 bytes long, for example, 0x1002 and 0x100A. Customers can use C51 interface or API to operate the R5FS.

2.8.1 ROCKEY5 TIME File Type

- **Executable File**

Executable files are programmed in C51 and be executed inside ROCKEY5 TIME. External programs call ROCKEY5 TIME API to pass parameters to executable file which runs inside ROCKEY5 TIME and will return results to external programs. Executable files can be modified by other executable files. Software manufacturers can create and write executable files after verifying Developer Password. The content of executable files cannot be read by anyone. Remote update function can also modify internal executable files. Please refer to section 3.2.7.

- **Internal Data File**

Internal Data File is used to store user data. Software manufacturers can use API to write data file but cannot read it.   Another way is using other internal executable files to read and write Internal Data File. That is application call internal execuable file over API to read and write Internal Data File.

- **Key File**

Key File is used to store RSA key pairs (public key and private key). Customers with Developer Password can write public keys. Internal executable files can read and write public keys. For private keys, customers with Developer Password can write. But nobody can read private keys. Use ROCKEY5 TIME utilities to generate RSA key pairs. Please refer section to 4.2.

File types and privilege：

		Developer Password	Executable File
Executable File		Write	Write
Internal Data File		Write	Read/Write
Key File	Public key	Write	Read/Write
	Private key	Write	Write

Table 2-1 File types and privilege

2.9 Other Functions

ROCKEY5 TIME encrypts communication between USB and dongle.

ROCKEY5 TIME has hardware based RSA algorithm (512/1024/2048 bit). It also provides DES and 3DES encryption algorithms. Symmetri key is exchanged securily by using RSA algorithm. Please note: If use C51 to do RSA calculation, it accepts the length of RSA keys is 512, 1024 or 2048. If use API to do RSA calculation, it only accepts 1024 bits long. Please refer to section 6.4.11, 6.4.12, 6.4.13, 7.1.20 and 7.1.21.

ROCKEY5 TIME also provide MD5 and SHA-1 algorithms.

ROCKEY5 TIME provides AES encryption algorithm (128 bits).

Because ROCKEY5 TIME can do floating point and double precision floating point arithmetic, software manufacturers can burn some formulas into ROCKEY5 TIME. The floating point and double precision floating point arithmetics related to the formulas can be executed inside the hardware of ROCKEY TIME and the performace is better than in C51.

There are 3 status of LED light: on, off and blinking. Software manufacturers can use API to control the LED light to confuse illegal users. For example, the LED light blinking may not mean that the ROCKEY5 TIME is executing internal programs.

ROCKEY5 TIME provides remote update function. Software manufacturers use a tool(refer to section 4.2.6) or API （refer to section 7.1.28 and 7.1.29） to generate and send an update package to end users and end users use a corresponding tool to to update internal files(refer to section 3.2.7). In this way, ROCKEY5 TIME dongles needn’t to be sent back to software manufactures. Such an update tool is included in SDK.

Different internal executable files share the shared memory zone. They use shared memory zone to share data. This way is more efficiency and cost less hardware consumption.

Chapter 3. Software Protection Paradigms

In the previous chapter, we have introduced many functions of ROCKEY5 TIME. How can these functions be used to protect applications? This chapter will introduce some software protection techniques and paradigms. All protection methods in this chapter can be found on the Internet or from books. So the best software protection solution is that customers design their own unique protection method.

3.1 Some Simple Skills

For software protection, knowing assembly language or system knowledge is not a necessary requirement. Whatever the software protection skill is chosen, the key point is to check some conditions. If the condition is met, programs can be allowed to continue running. Otherwise, report some messages and exit. Whatever customers try to remove these codes, these codes are still in programs. Hence what we need to do is trying to hide or protect these codes.

3.1.1 Confused Code

Generally speaking, programmers and crackers believe that every piece of code in software must be useful. They believe no junk code in software. Crackers usually trace every code and try to understand what this code does. This method intentionally adds lots of junk code to make the software more and more complex. Lots of junk codes will confuse crackers and waste lots their time on analyzing junk code.

3.1.2 Fake Checks

In software, intentionally check conditions with some incorrect values. If check result is correct, it means that the software is being cracked. Then try to exit programs or go on confusing, etc.

3.1.3 Randomize Check Points

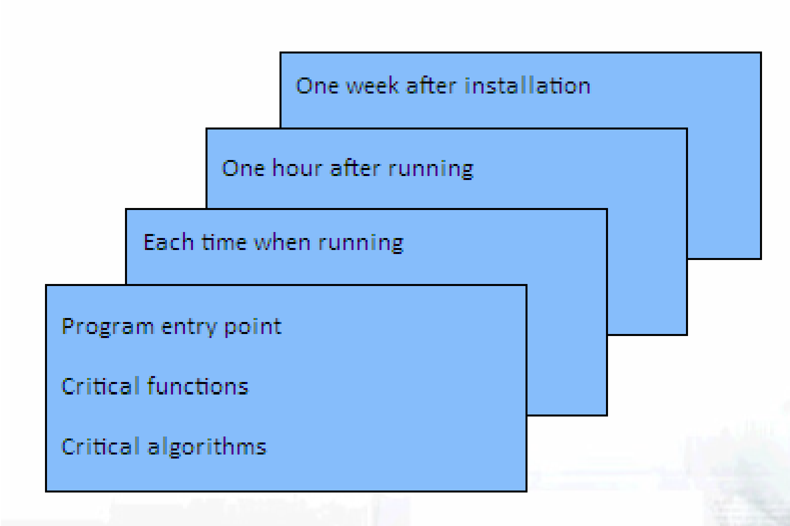


Figure 3.1.3-1 The Time of Check Points

Try to randomize check points in software. Finding all check points is not an easy work for crackers.

3.2 Use ROCKEY5 TIME to Encrypt Software

Customers had better design their own software protection solution. Do not share with others or use well-known solutions. Following samples show how to use ROCKEY5 TIME to protect software.

3.2.1 New Methods of ROCKEY5 TIME

Many traditional software protection methods use software protection dongles just as storage devices. Put some data into dongle and read out for checking when needed. There are also many developers only detect the existence of hardware, such encryption ways are very easy to be cracked. Later software protection dongles allow writing user defined algorithm. However, due to hardware limitation, imported algorithms only accept addition, subtraction, left rotation, right rotation calculations, etc. The most important thing is it cannot implement specific function of software.

Due to the smart card technology involved, ROCKEY5 TIME can be used to implement some functions that only PC can support. Software manufacturers could follow several steps to encrypt their software.

(1) Use high level programming language to develop software.

(2) Convert critical algorithms into C51 and build binaries. Put converted binaries into ROCKEY5 TIME.

(3) Converted binaries are running inside ROCKEY5 TIME. External programs pass arguments into ROCKEY5 TIME and receive results from ROCKEY5 TIME.

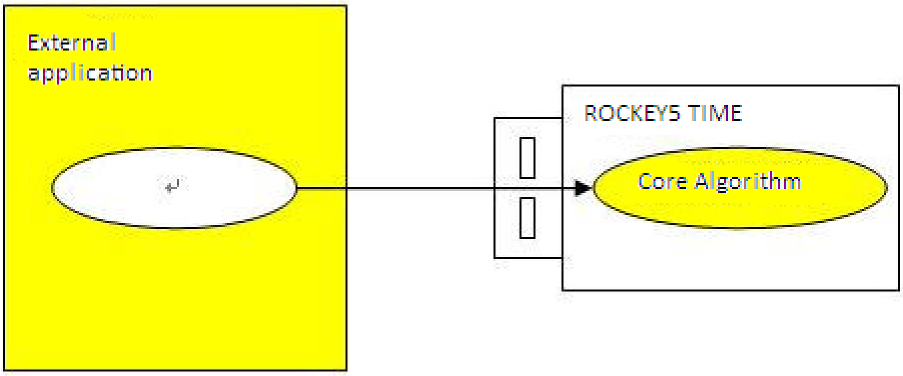


Figure 3.2.1-1 ROCKEY5 TIME Scheme

Lots of business softwares, such as CAD software, project budget software and financial software, contain lots of mathematical formulas. It is very easy to protect these kinds of software with ROCKEY5 TIME. ROCKEY5 TIME has capacity of calculating double-precision floating point, so it can calculate complex trigonometric functions. Based on this feature, algorithms, which can be only run in PC before, now can be run inside ROCKEY5 TIME. Crackers could only get the input and output but cannot know what was done inside ROCKEY5 TIME. Without attaching ROCKEY5 TIME, the calculation is not correct so that the software cannot continue. Please note:

(1) The algorithm should be complex enough. If the algorithm is too simple, for instance, an addition calculation, crackers can guess it.

(2) As ROCKEY5 TIME is 32 bit smart card chip based, it is fast enough for normal calculation. If the algorithm is very complex, customers can put part of the algorithm inside ROCKEY5 TIME.

(3) The algorithm should not be an open algorithm or leaked out algorithm.

(4) The returned result of internal algorithm should be taken part in calculation. Do not use it for determine conditions. Determine position is very easy to be found by cracker.

Above all are the most efficient ways to protect software. Use ROCKEY5 TIME as a black box and pay more attention on protection skills. If so the software will not be cracked.

3.2.2 Check Logic inside ROCKEY5 TIME

Software manufacturers can use API or C51 interface to get hardware information, such as HID or VID. As hardware information is unique, software manufacturers should check them inside ROCKEY5 TIME to avoid being cracked. Adding some random numbers into software will greatly increase the software complexity. For example, firstly generate a random number. Then Process hardware information with the random number. Pass the result into ROCKEY5 TIME. In ROCKE5 TIME, work out the hardware information and compare it with hardware information which is obtained by C51 interface. Cracker will only get some random values since an algorithm is inside dongle. The software cannot be cracked.

3.2.3 Make Input/Output Maze

Some crackers use USB analyzer to catch transported data between dongles and PC. Even the communication between ROCKEY5 TIME and PC is encrypted, it is strongly recommended to add some customized security algorithms. For example, encrypt arguments before passing to internal programs. Internal programs firstly decrypt arguments then begin to calculate.

3.2.4 Build Application with 3DES and RSA

ROCKEY5 TIME uses 32 bit smart card chip, which provides high performance 3DES and RSA calculation capacity. 3DES and RSA can be used for C/S, B/S software as well as single version software.   Here take the C/S software for example:

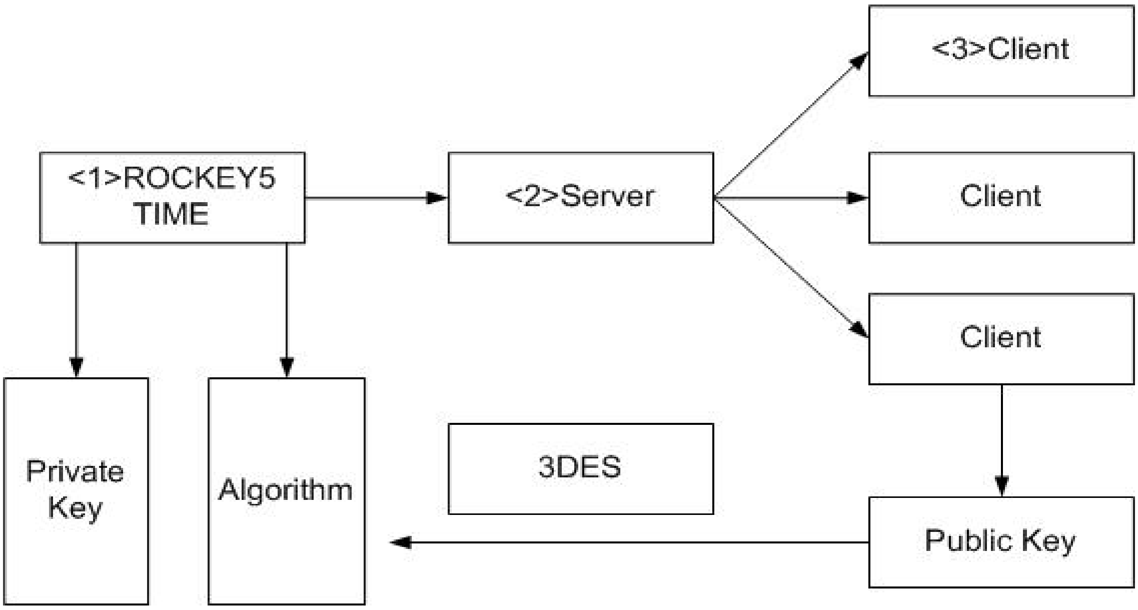


Figure 3.2.6-1   3DES and RSA

<1> ROCKEY5 TIME is attached on server. It contained user defined algorithm and RSA private key.

<2> Server application processes data which come from clients and then pass to ROCKEY5 TIME. After ROCKEY5 TIME finished calculating, server application transports data to client.

<3> Client application contained a RSA public key.

Client application generates a random number to use as the key of 3DES. Then client uses public key to encrypt the 3DES key and transports to server application. Server application passes this data into ROCKEY5 TIME. ROCKEY5 TIME uses private key to decrypt the data and store the 3DES key as an internal file. After this, all communication data will be encrypted by this 3DES key.

For some experienced crackers, they can modify the client application. For example, a cracker buys a ROCKEY5 TIME and writes his private key into in. Then he uses his own public key to replace the one which is stored in client application. So software manufacturers had better to add some specific algorithms into ROCKEY5 TIME so that even the private key/public key has been replaced, crackers still cannot crack the software. Two things need to be considered: 1. Inside the ROCKEY5 TIME, there must be an algorithm that relate with key pairs. If the key pair is replaced, the algorithm returns wrong result. 2. All check points should be placed into ROCKEY5 TIME. Server application is only used for transportation.

3.2.5 Remote Update

Software manufacturers could use remote update function to update ROCKEY5 TIME contents for end users. Following parts can be updated: 8KB Data Memory, 8 pairs RSA keys, 8 3DES keys, 8 AES keys, expiry date and total use hours. 32KB user memory cannot be updated. It needs users to modify by them.

Please note: The update package for updating expiry date or total use time can be only updated once. There is an UTC time stamp inside the update package. If the time stamp is invalid, ROCKEY5 TIME refuses to update.

Update process:

(1) When released protected software, ROCKEY5 TIME stores internal executable files, internal data and another executable file that is used for update. Here we called this file as UP file. When executing update, keep all transmitted data encrypted. We use RSA to encrypt data so the private key is needed to store inside ROCKEY5 TIME. For example, inside the ROCKEY5 TIME memory there are several files: file A (executable file), file B (internal data), PRI (private key) and a UP (executable file) file.

(2) When software is updated, software manufacturers need to encrypt new file A' and new file B' with public key. After this, send the encrypted package to end users. Please note: Public key should not be stored inside ROCKEY5 TIME due to security issue.

(3) When end users received update package, use UP file to decrypt A' and B' with private key. Then use C51 file operation interface to replace old A and B.

Update package can bind hardware ID so that it is only for specified ROCKEY5 TIME.

3.2.6 Time Management

Time Management is new feature of ROCKEY5 TIME. With Developer Password, customers can set expiry date of COS. Time Management function provides two types of time limitation methods:

（1） Set software expiry date

Software manufacturers set an expiry date (UTC time) into ROCKEY5 TIME. If the expiry date is reached, the decryption function will be locked and software cannot be executed anymore.

（2） Set software total use hours

Software manufacturers set total user hours into ROCKEY5 TIME. It records the elapsed time when software is running. If the elapsed time is reached total hours, the decryption function will be locked and software cannot be executed anymore.

Here are functions (APIs) that will be locked when software expired.

API			
API function	API function name	API function section	API function

Section			name
7.1.22	RY5_PublicEncrypt	7.1.27	RY5_3DES
7.1.23	RY5_PrivateDecrypt	7.1.30	RY5_AES
7.1.24	RY5_MD5	7.1.5	RY5_GetRandom
7.1.25	RY5_SHA1		

Table 3.2.6-1

Chapter 4. ROCKEY5 TIME Utilities

Previous chapter demonstrate a whole process of using ROCKEY5 TIME. In this chapter, we will introduce the ROCKEY5 TIME utilities, such as KEIL IDE, ROCKEY5 TIME User Tool, ROCKEY5 TIME Virtual File System Manager and ROCKEY5 TIME Envelope.

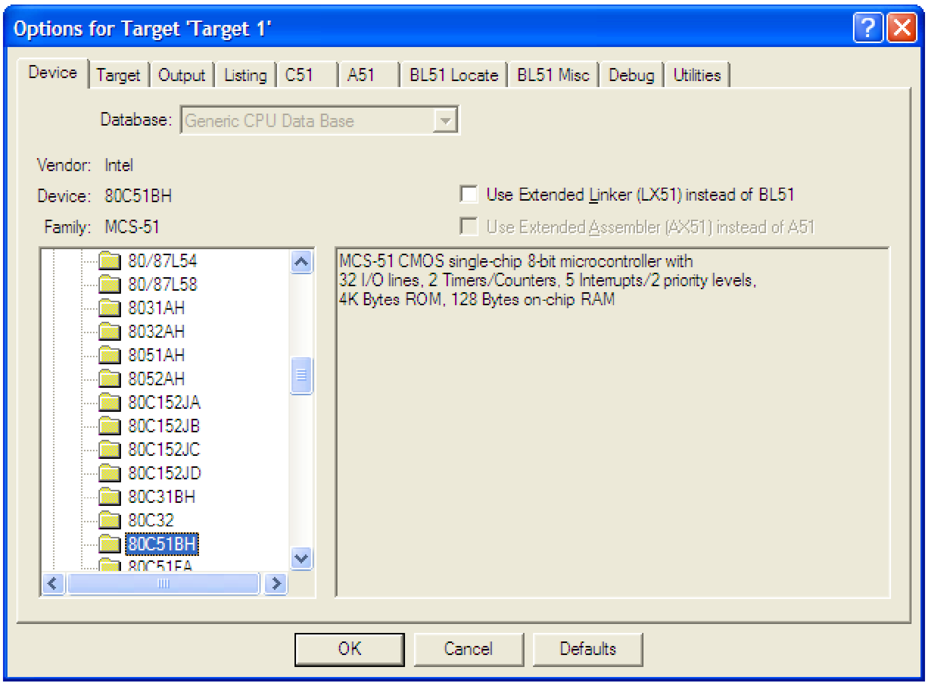
4.1 KEIL IDE

The internal executable programs of ROCKEY5 TIME are written in C51. The grammar of C51 and C language is basically same. C51 is mostly used in hardware programming field. KEIL IDE compiles C51 source code and builds to bin file. Bin files can be imported into ROCKEY5 TIME and be executed by calling API. Please refer section 7.1.15.

C51 IDE from KEIL Software Company is named uVision. Customers can download the trail version from KEIL website (http://www.keil.com).

4.1.1 Create a Project

To create a new project, open “Project” of KEIL uVision2 and click “New Project”. Input the project name in the pop-up dialog box and save it. Choose 51 serial CPU when “Options for Target ‘Target1’” appears. Users need to re-choose CPU for the existing project without 51 serial CPU. Please see this figure:


Options for Target 'Target 1'

Database: Generic CPU Data Base

Vendor: Intel

Device: 80C51BH

Family: MCS-51

□ Use Extended Linker (LX51) instead of BL51

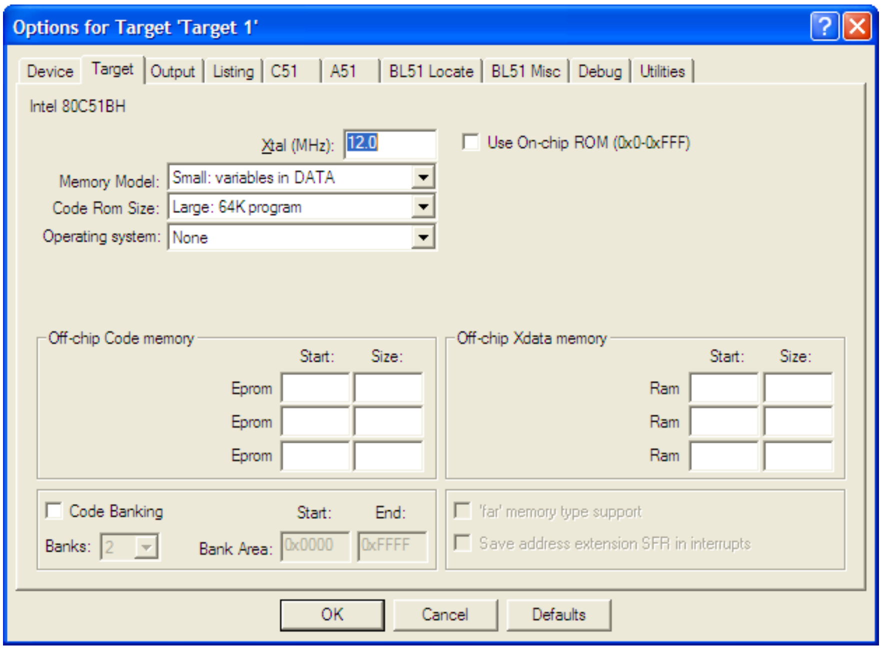
□ Use Extended Assembler (AX51) instead of A51

MCS-51 CMOS single-chip 8-bit microcontroller with
32 I/O lines, 2 Timers/Counters, 5 Interrupts/2 priority levels,
4K Bytes ROM, 128 Bytes on-chip RAM

Figure 4.1.1-1 Select 51 serial CPU

4.1.2 Set Project Options

Click “Project -> “Options for Target ‘Target1’” to get high efficiency and use default setting (small model) in Target menu. Please see the figure below:



Intel 80C51BH

Xtal (MHz): 12.0

□ Use On-chip ROM (0x0-0xFFF)

Memory Model: Small: variables in DATA

Code Rom Size: Large: 64K program

Operating system: None

Off-chip Code memory — Start: Size: Eprom, Eprom, Eprom

Off-chip Xdata memory — Start: Size: Ram, Ram, Ram

□ Code Banking — Start: End:

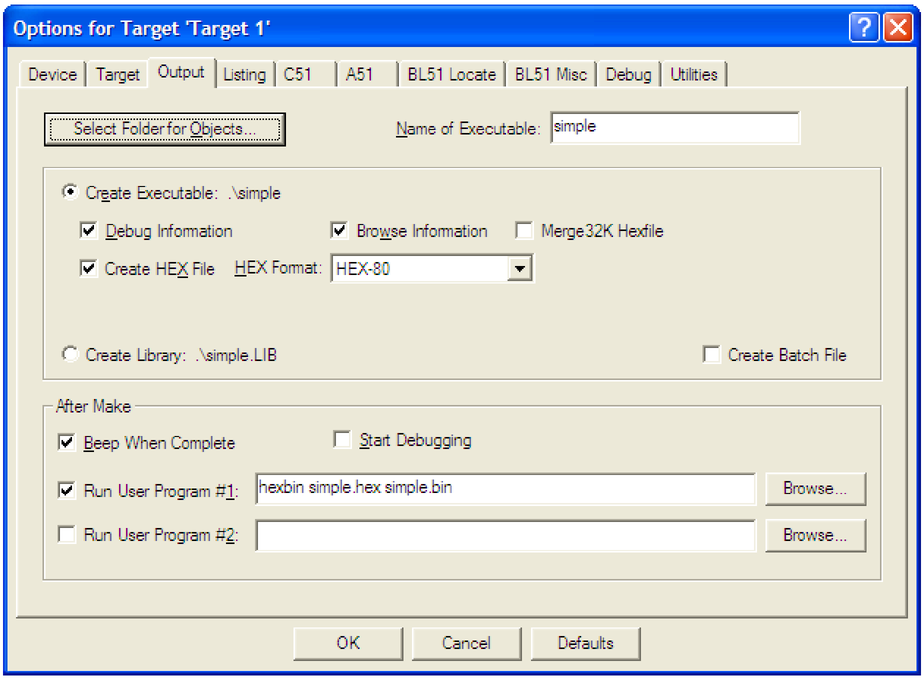
Banks: 2 — Bank Area: 0x0000 0xFFFF

□ 'far' memory type support

□ Save address extension SFR in interrupts

Figure 4.1.2-1 Choose Small Model

In “Options for Target ‘Target 1’” page, click “Output” page and tick both “Create HEX File” and “Run User Program #1”. Input “hexbin.exe test.hex test.bin” in the text box. The name “test.hex” and “test.bin” will be changed according to the name of the project. Please copy hexbin.exe (in Utilities folder) to the project folder. Hexbin.exe file is used to transfer the hexadecimal files into binary files. Binary files are used to be imported into ROCKEY5 TIME. Please see the figure below:



Name of Executable: simple

● Create Executable: .\simple

☑ Debug Information ☑ Browse Information □ Merge32K Hexfile

☑ Create HEX File HEX Format: HEX-80

○ Create Library: .\simple.LIB □ Create Batch File

After Make

☑ Beep When Complete □ Start Debugging

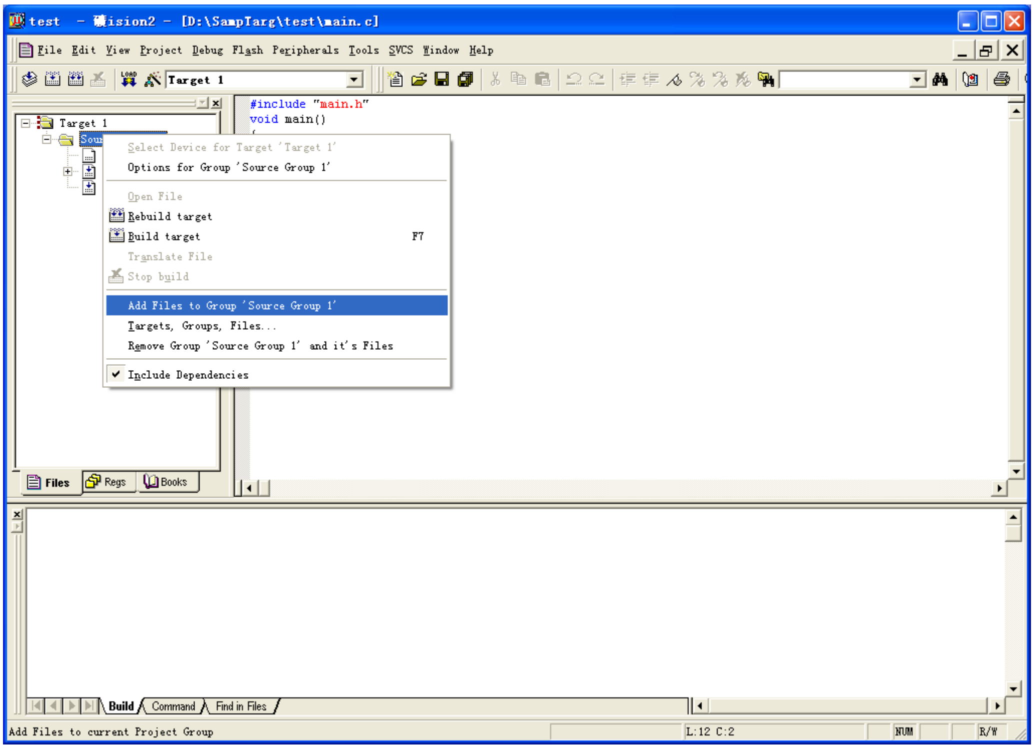
☑ Run User Program #1: hexbin simple.hex simple.bin

□ Run User Program #2:

Figure 4.1.2-2 Output Options

After finished all configuration steps, select “View” -> “Project window” to spread the tree and right click “Source

Group 1”. Select “Add files to Group ‘Source Group 1’” to add ROCKEY5.h and small_model.LIB. Last, add C51 source code file into the project. Please see the figure below:


test - µVision2 - [D:\SampTarg\test\main.c]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Target 1
Sou
Select Device for Target 'Target 1'
Options for Group 'Source Group 1'
Open File
Rebuild target
Build target F7
Translate File
Stop build
Add Files to Group 'Source Group 1'
Targets, Groups, Files...
Remove Group 'Source Group 1' and it's Files
Include Dependencies
Files
Regs
Books
Build
Command
Find in Files
Add Files to current Project Group
L:12 C:2
NUM
R/W

```
#include "main.h"
void main()
{
```

Figure 4.1.2-3 Add files

4.1.3 Debug

After finished above processes, programs move to Debug mode. About KEIL uVision2 debugger, please read its user manual.

4.1.4 Exit

To end the debugging process, “_exit()” has to be used at the end of the debugging program. Please refer section 6.1.1. Users could also select “Debug -> Start/Stop Debug Session” in menu to stop debugging process.

C51 programs will be terminated when “_exit()” is called. Users can add “_exit()” to anywhere in C51 program to check the intermediate program value.

4.1.5 Import C51 Programs

C51 programs can be written into real smart card or virtual smart card. Use ROCKEY5 TIME Virtual File System Manager to import BIN files into virtual smart card. Please refer section 4.3.4. Use ROCKEY5 TIME User Tool to import BIN files into real smart card. Please refer section 4.2.5. Or use API to import. Please refer section 7.1.16 and 7.1.17.

After imported, use ROCKEY5 TIME User Tool or API to execute C51 programs to check result. Please refer section 4.2.3 and 7.1.18.

4.2 ROCKEY5 TIME User Tool

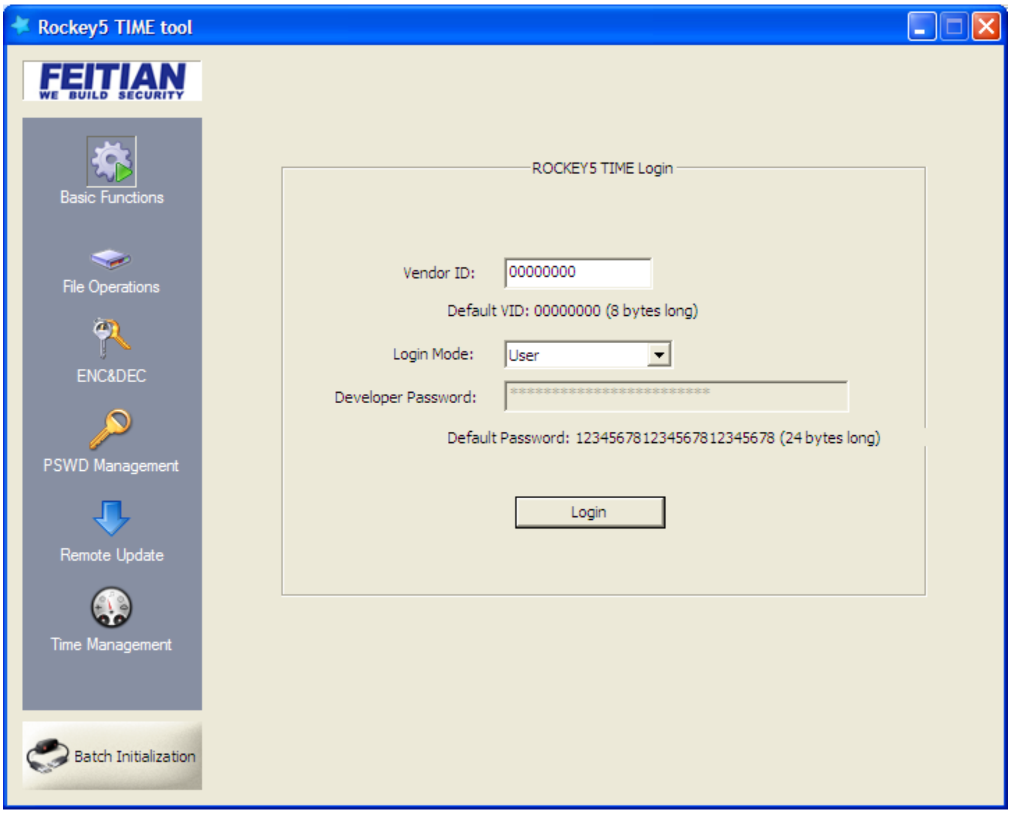
ROCKEY5 TIME User Tool provides:

- Basic Functions: access Data Memory zone and shared Memory zone; generate random number; control LED light.
- File Operations: create, write, execute and clean files.
- Encryption and Decryption: perform encryption and decryption and generate RSA key pairs.
- Password Management: get hardware ID, change Developer Password and set Vendor ID.
- Remote Update: generate remote update package and test remote update package.
- Time Management: get current COS expiration dates and set COS expiration dates.
- Batch Initialization: Initialize dongles.

Under Utilities folder, RY5Tools.exe is ROCKEY5 TIME User Tool.

4.2.1 Start and Login

Run RY5Tools.exe tool and you'll see the dialog as the figure shown below.



ROCKEY5 TIME Login

Vendor ID: 00000000

Default VID: 00000000 (8 bytes long)

Login Mode: User

Developer Password: ************************

Default Password: 123456781234567812345678 (24 bytes long)

Figure 4.2.1-1 ROCKEY5 TIME User Tool

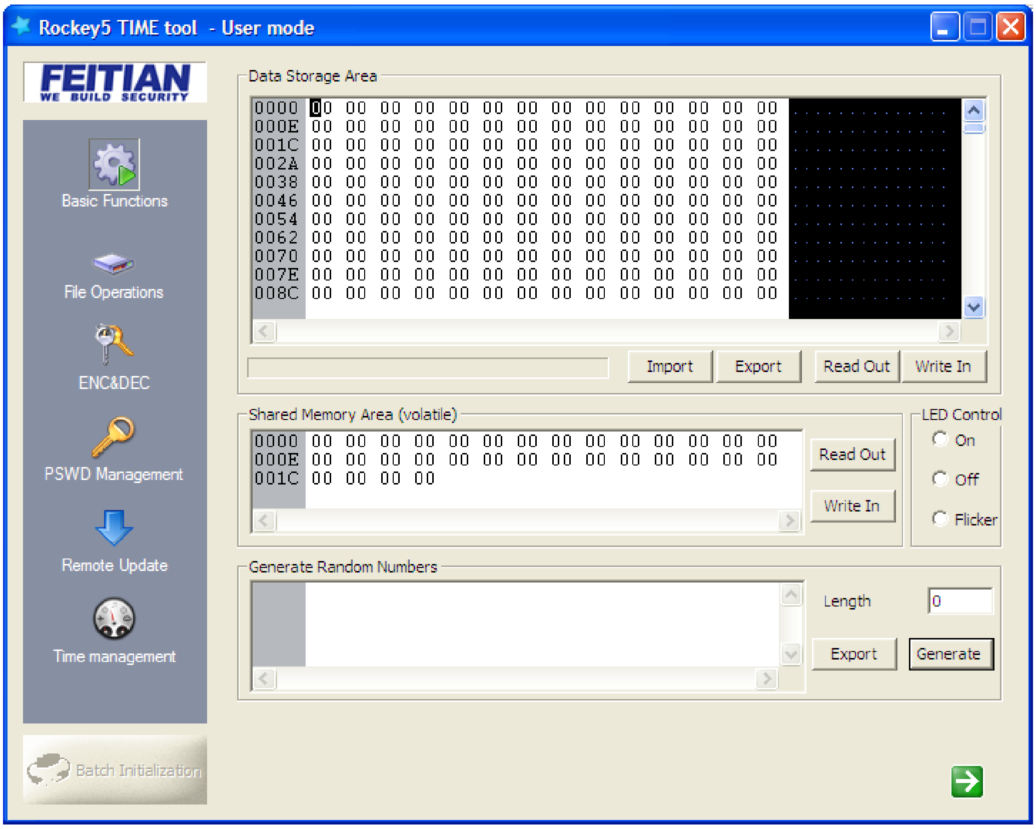
This tool only supports one ROCKEY5 TIME dongle. Please attach a ROCKEY5 TIME dongle to PC before running this tool. If there are more than one ROCKEY5 TIME dongles, this tool will operate the first one.

When using this tool, removing ROCKEY5 TIME dongle will lead the tool to go back to login dialog.

Input Vendor ID and Developer Password, choose Login Mode. There are two Login Modes. If selected User Mode, Developer Password is not necessary to input. Some functions relevant with Developer Password cannot be used with User Mode. Click “Login” button to continue.

4.2.2 Basic Functions

After successfully login, ROCKEY5 TIME User Tool will enter Basic Functions interface. Please see the figure below:



Data Storage Area

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
001C 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0046 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0062 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00
007E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
008C 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Shared Memory Area (volatile)

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
001C 00 00 00 00
```

Length: 0

Figure 4.2.2-1 Basic Functions

■ Data Storage Area

Users can directly edit data in data display area. Click “Write in” button to save into Data Storage Area. If failed to write, users need to check errors based on error code. Please refer section 7.2.1.

Click “Read out” to display data which are stored in Data Storage Area. Data will be displayed in data display area.

Click “Import” button and select a file in pop up dialog. Data stored in the file will be displayed in data display area. Then click “Write in” button to write them into Data Storage Area.

Click “Export” button to save data into a file.

■ Shared Memory Area

Users can directly edit data in data display area. Click “Write in” button to save data into Shared Memory Area. If failed to write, users need to check errors based on error code. Please refer section 7.2.1.

Click “Read out” to display data which are stored in Shared Memory Area. Data will be displayed in data display area.

■ LED Control

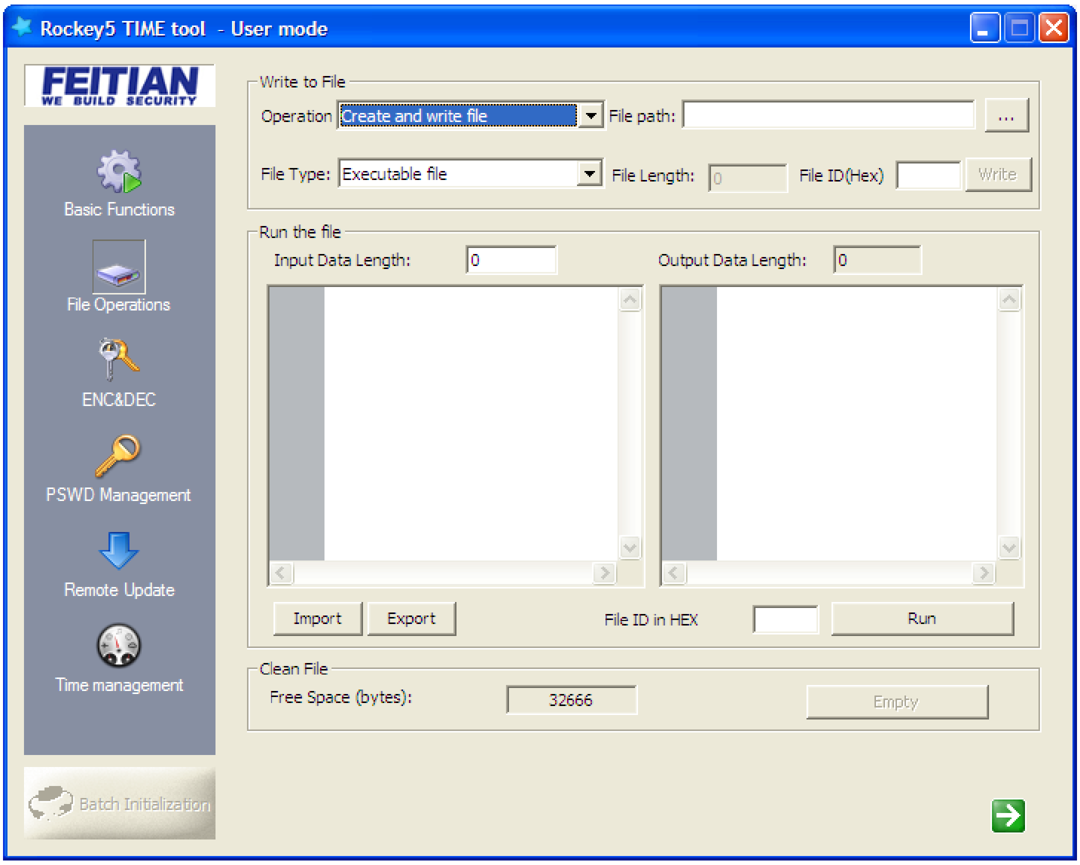
Select “On”, “Off” or “Flicker” to control ROCKEY5 TIME’s LED light.

■ Generate Random Number

Input the length of desired random number and click”Generate” button to get a random number. Click “Export” button to save this number into a file.

4.2.3 File Operation

Click “File Operations” button to enter “File Operations” interface. Please see figure below:



Rockey5 TIME tool - User mode

Write to File

Operation: Create and write file

File path:

File Type: Executable file

File Length: 0

File ID(Hex):

Run the file

Input Data Length: 0

Output Data Length: 0

File ID in HEX

Clean File

Free Space (bytes): 32666

Figure 4.2.3-1 File Operations

■ Write to File

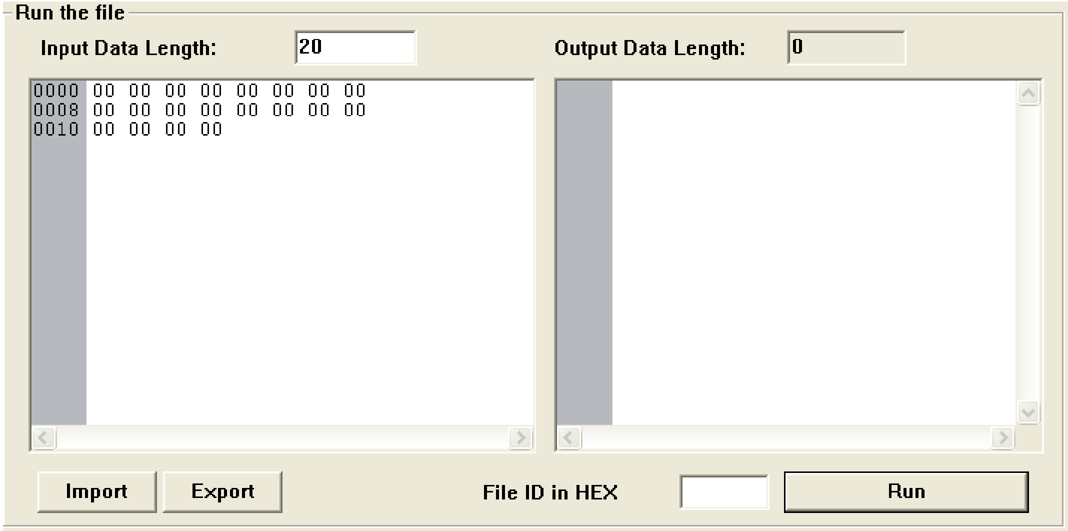
Select “Operation”, “File Path”, “File Type” and input file size and file ID one by one. Then click “Write” button to write files into ROCKEY5 TIME. If failed to write, users need to check errors based on error code. Please refer section 7.2.1.

Operation: create file; write file; create empty file; import file from Virtual File System.

File Type: executable file, data file, public key file and private key file. Please refer section 2.7.1.

■ Execute a File

Only executable files can be executed. At first, input the length of input data. Then go to the data display area to input data. Or click "Import" button to input data from a file. Please see figure 4.2.3-2. Input executable file ID then click "Run" button to execute. The result will be displayed in data display area. If failed to run, users need to check errors based on error code. Please refer section 7.2.1.



Run the file

Input Data Length: 20

Output Data Length: 0

```
0000 00 00 00 00 00 00 00 00
0008 00 00 00 00 00 00 00 00
0010 00 00 00 00
```

File ID in HEX

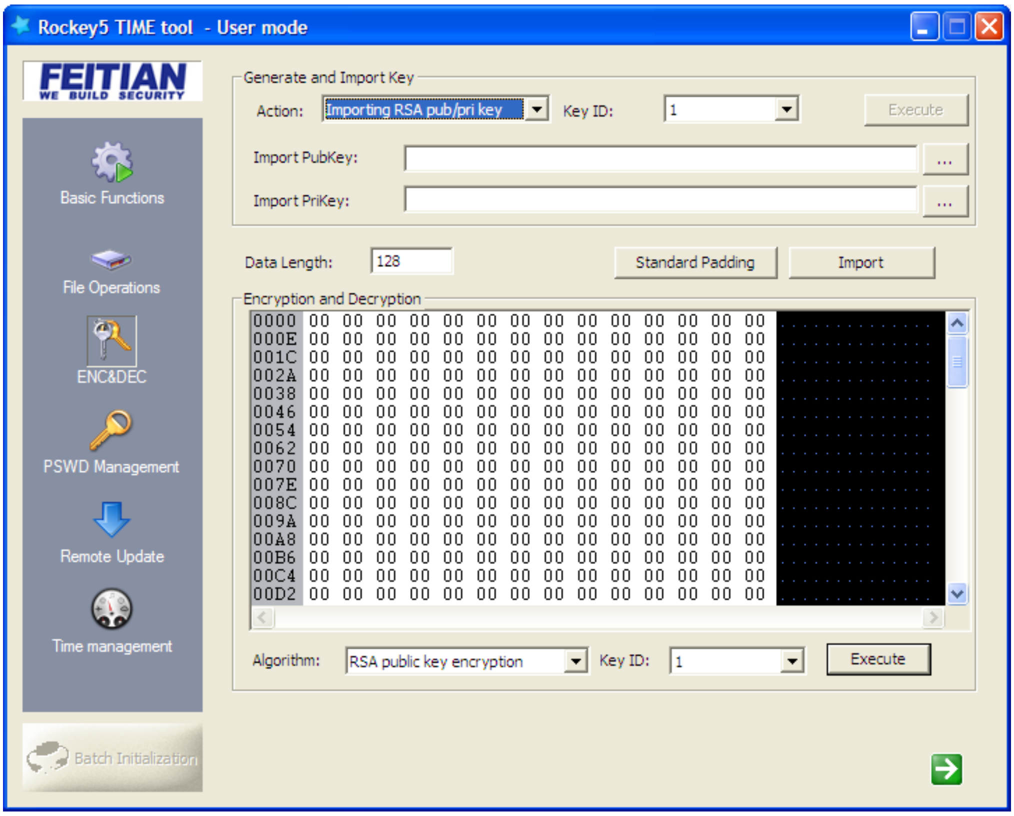
Figure 4.2.3-2 Run a File

■ Clean File

Remove all files in ROCKEY5 TIME. Click "Empty" button to remove all files in ROCKEY5 TIME. If failed to clean, users need to check errors based on error code. Please refer section 7.2.1.

4.2.4 Encryption and Decryption

User Tool enables users to import/export RSA key pairs, 3DES key and AES key. It can also generate RSA key pair and export it out of ROCKEY5 TIME for backup. ROCKEY5 TIME can store up to 8 key pairs and one remote update key. If customer wants to use RSA in software, he or she needs to generate a RSA key pair firstly. Users can use C51 to generate RSA key pair. But in this way, the private key cannot be exported out of ROCKEY5 TIME. In this case, each ROCKEY5 TIME contains a unique private key. If customer wants to share a private key with many ROCKEY5 TIME, use User Tool or API to generate RSA key pair then save them into files (Please refer section 7.1.20). Then use User Tool, Virtual File Manager or API to import key pair file into other ROCKEY5 TIME. Click "ENC & DEC" button in User Tool dialog, the Encryption and Decryption interface will show as below:



Generate and Import Key

Action: Importing RSA pub/pri key Key ID: 1 Execute

Import PubKey: ...

Import PriKey: ...

Data Length: 128 Standard Padding Import

Encryption and Decryption

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
001C 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0046 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0062 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00
007E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
008C 00 00 00 00 00 00 00 00 00 00 00 00 00 00
009A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A8 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B6 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C4 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D2 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Algorithm: RSA public key encryption Key ID: 1 Execute

Figure 4.2.4-1 Enc and Dec interface

■ Generate and import key pair

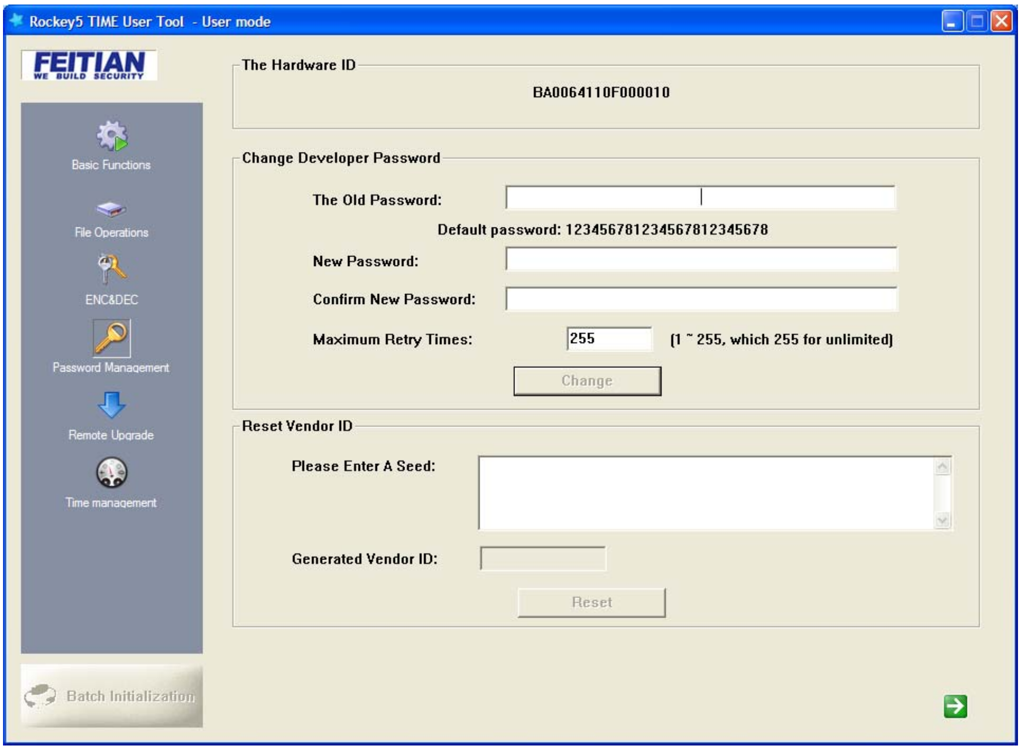
Customers can generate a key pair or import an existed key pair into ROCKEY5 TIME. Select "Operation" and input the key number. Then specify a file to be written for generated key pair or a file that contains existed key pair. Click "Execute" button to generate or import. If failed to run, users need to check errors based on error code. Please refer section 7.2.1.

■ Encryption and Decryption

Input data length and directly edit data display area. Or click "Standard Padding" to automatically write data. Or import data from a file. After this, select an algorithm and specify a key to perform encryption/decryption. If failed to run, users need to check errors based on error code. Please refer section 7.2.1.

4.2.5 Password Management

Click "Password Management" to open the "Password Management" interface. Please see the figure below:



Rockey5 TIME User Tool - User mode

The Hardware ID

BA0064110F000010

Change Developer Password

The Old Password:

Default password: 123456781234567812345678

New Password:

Confirm New Password:

Maximum Retry Times: 255 (1 ~ 255, which 255 for unlimited)

Reset Vendor ID

Please Enter A Seed:

Generated Vendor ID:

Figure 4.2.5-1 Password Management

- Change Developer Password

Input old password, new password twice and click “Change” button to change Developer Password. If failed to change, users need to check errors based on error code. Please refer section 7.2.1.

- Reset Vendor ID

Input a seed and click “Reset” button to get a new Vendor ID. If failed to set, users need to check errors based on error code. Please refer section 7.2.1.

4.2.6 Remote Update

For detailed information about remote update, please refer section 3.2.5. This section will introduce how to use User Tool to update remotely.

Click “Remote Update” button to show the following figure:

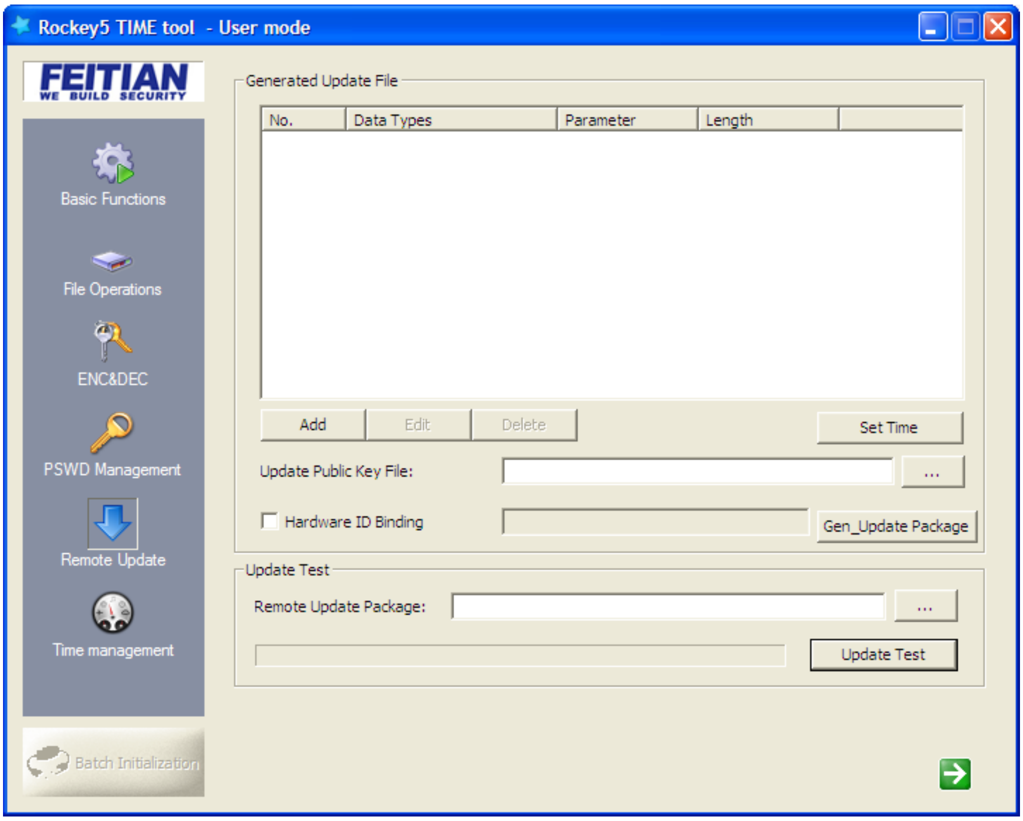
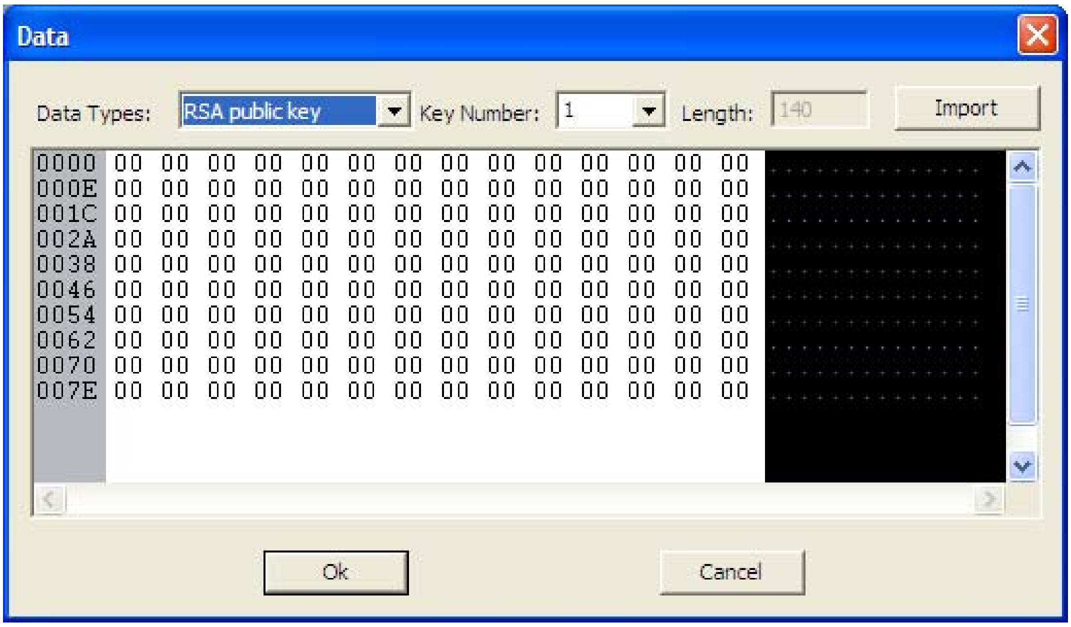


Figure 4..6-1 Remote Update

■ Generate Update Package

Click “Add” button and fill update data.

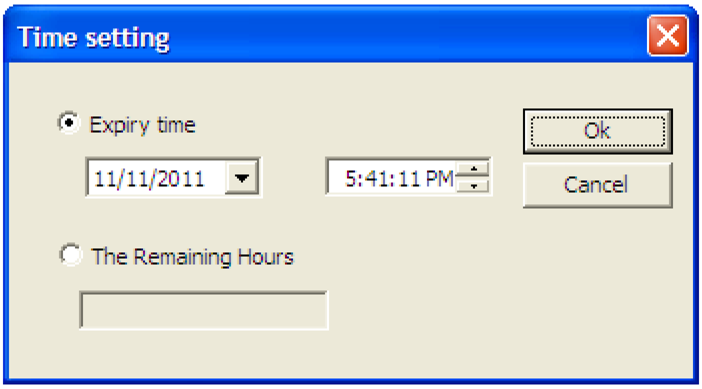


```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
001C 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0046 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0062 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00
007E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 4.2.6-2 Add data

Select “Data Type” and input data or import data from a file. Click “OK” to go back to Remote Update interface.

Click “Set Time” button to set expiration date or remaining hours. Please see the figure below:



Time setting

● Expiry time

11/11/2011

5:41:11 PM

Ok

Cancel

○ The Remaining Hours

Figure 4.2.6-3 Set Time

After set time, click “OK” button to continue. Customers can edit or delete existed update packages by clicking “Edit” and “Delete” buttons.

Click “...” button and choose public key file.

If customer inputs a hardware ID in HID binding area, the update package will be only for the specified ROCKEY5 TIME dongle whose HID is the same as you input.

Input a file name and click “Generate Update Package” button to finish generating update package.

■ Update Test

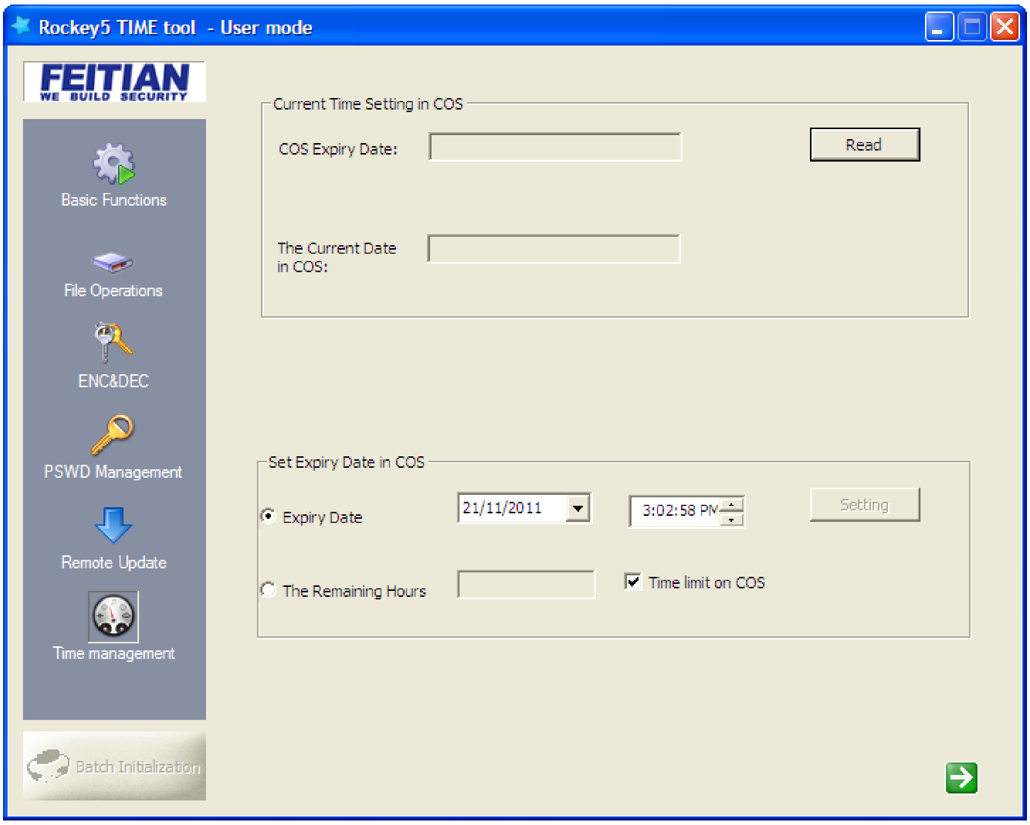
Click “...” button and select an update package. Then click “Update Test” button to test. If failed to test, users need to check errors based on error code. Please refer section 7.2.1.

Due to ROCKEY5 TIME needs private key to decrypt update package, customers must import private key before updating. Please refer section 4.2.4. Public key is not recommended to be imported in dongle.

4.2.7 Time Management

For detailed information about Time Management, please refer section 3.2.6. This section will introduce how to use Time Management in User Tool.

Click “Time Management” button to show the following figure:



Current Time Setting in COS

COS Expiry Date:

The Current Date in COS:

Set Expiry Date in COS

● Expiry Date 21/11/2011 3:02:58 PM

○ The Remaining Hours

☑ Time limit on COS

Figure 4.2.7-1 Time Management

■ Current Time Setting

Click “Read” button to get the current time setting in COS.

■ Set Expiration Date

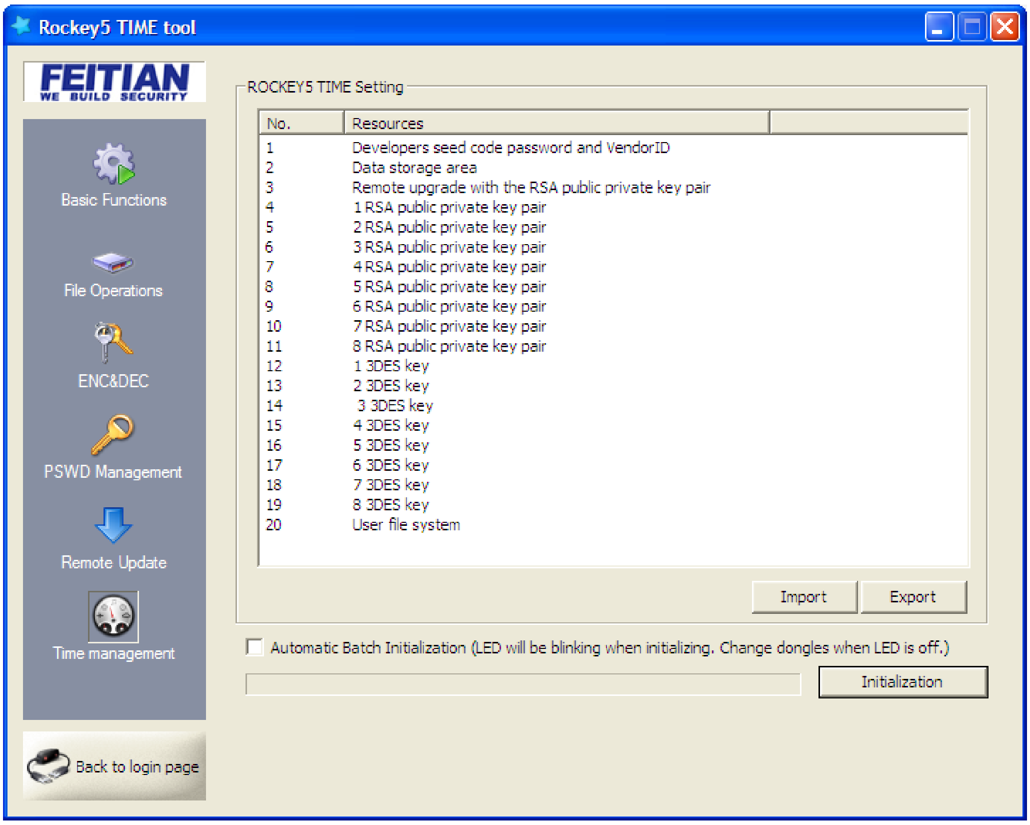
Customers could set COS expiration date or remaining hours. Click “Setting” button to finish setting. If failed to set, users need to check errors based on error code. Please refer section 7.2.1.

Note: Only Developer Mode supports setting time in COS.

4.2.8 Initialization

Software can use Bath Initialization function to initialize ROCKEY5 TIME in batch.

Click “Batch Initialization” button to show following figure:



ROCKEY5 TIME Setting

No.	Resources
1	Developers seed code password and VendorID
2	Data storage area
3	Remote upgrade with the RSA public private key pair
4	1 RSA public private key pair
5	2 RSA public private key pair
6	3 RSA public private key pair
7	4 RSA public private key pair
8	5 RSA public private key pair
9	6 RSA public private key pair
10	7 RSA public private key pair
11	8 RSA public private key pair
12	1 3DES key
13	2 3DES key
14	3 3DES key
15	4 3DES key
16	5 3DES key
17	6 3DES key
18	7 3DES key
19	8 3DES key
20	User file system

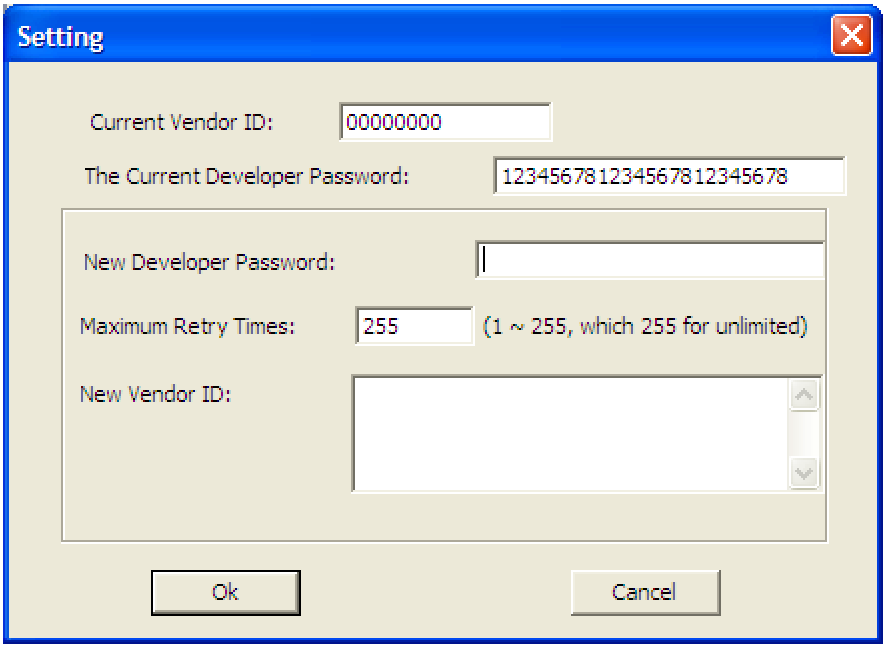
□ Automatic Batch Initialization (LED will be blinking when initializing. Change dongles when LED is off.)

Figure 4.2.8-1 Initialization

ROCKEY5 TIME setting panel shows all items that can be initialized.

(1) Developer Password and Vendor ID

Double click the first item “1 Developer seed code Password and Vendor ID” to show the following figure:



Current Vendor ID: 00000000

The Current Developer Password: 123456781234567812345678

New Developer Password:

Maximum Retry Times: 255 (1 ~ 255, which 255 for unlimited)

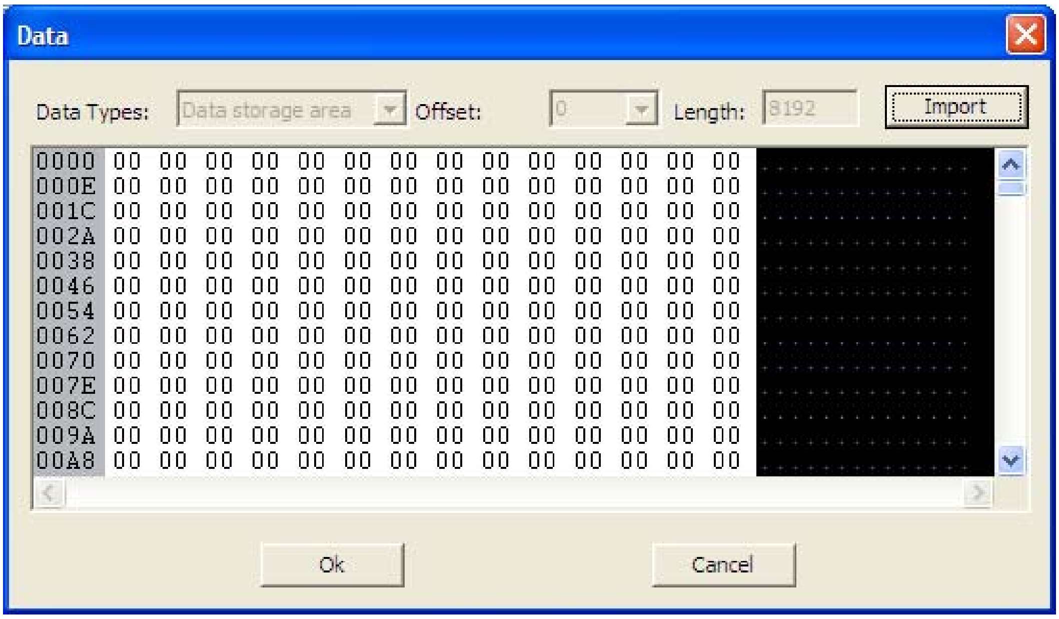
New Vendor ID:

Figure 4.2.8-2 Developer Password and Vendor ID

The default Vendor ID is 00000000 and the default Developer Password is 123456781234567812345678. After input new Developer Password, click “OK” button and the User Tool will find ROCKEY5 TIME with the current Vendor ID. If no ROCKEY5 TIME is found, User Tool will notify “waiting for attaching ROCKEY5 TIME”. Current Developer Password will be used to verify login. If failed to verify, User Tool will notify “Failed to verify Developer Password, error code is 0xF0000CFF”. Please note: The default retry time is unlimited. If customer has set a retry time, please input Developer Password carefully. Once the retry time is reached, ROCKEY5 TIME will be locked. In this case, the only way to unlock ROCKEY5 TIME is to send back to FEITIAN. Please refer section 2.2.

(2) Data Storage Area

Double click “2 Data Storage Area” will show the following figure:



Data

Data Types: Data storage area Offset: 0 Length: 8192

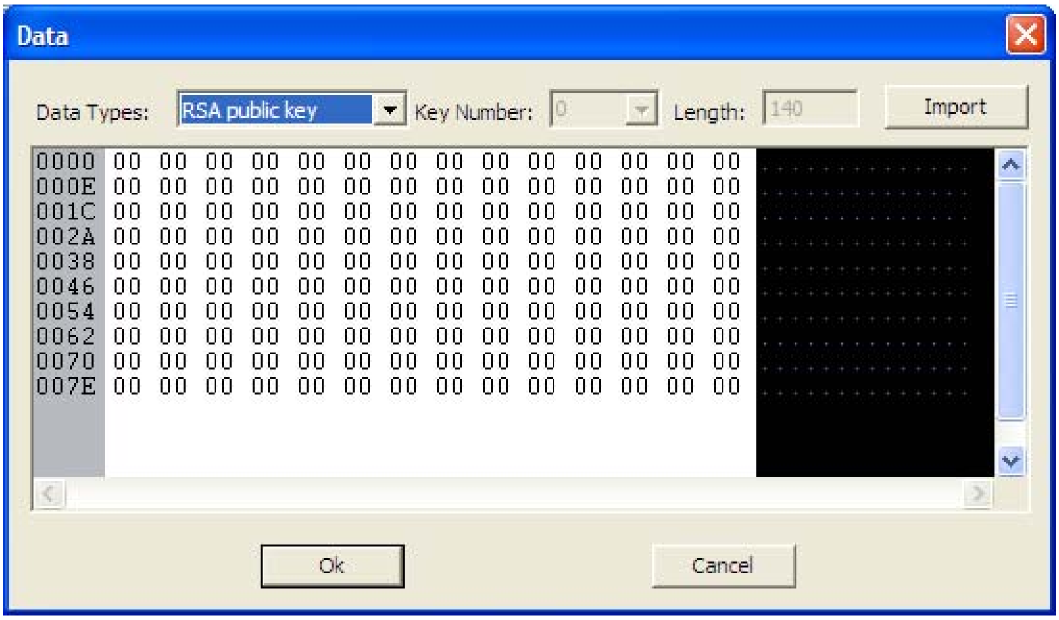
```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
000E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
001C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
002A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0038 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0046 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0054 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0062 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
007E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
008C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
009A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
00A8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
```

Figure 4.2.8-3 Data Storage Area

Customers can directly edit hexadecimal value or ASCII value.

(3) Key Paris for Remote Update

Double click “3 Key Pairs for Remote Update” to show the following figure:

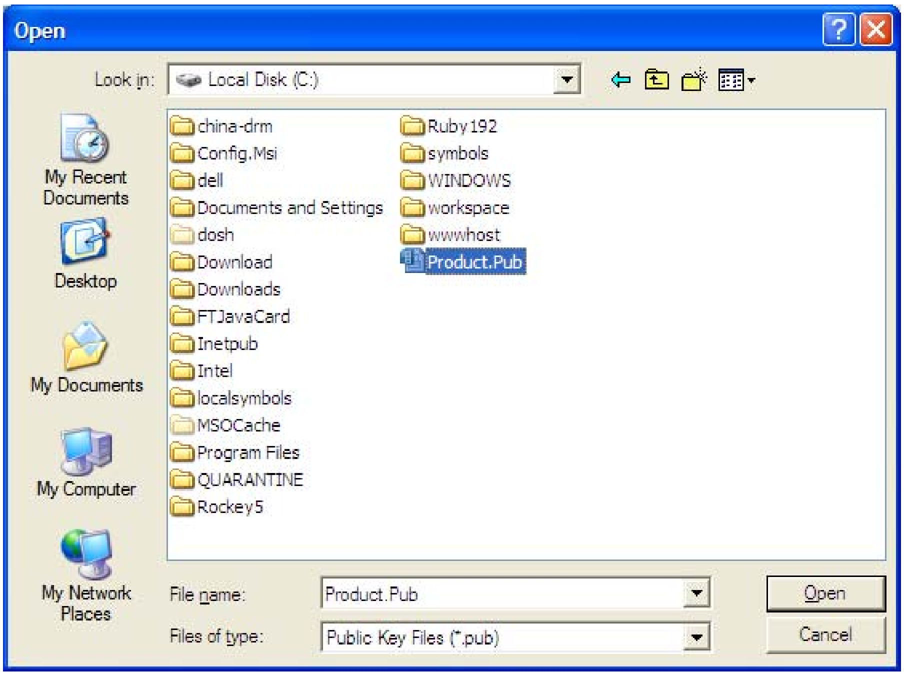


Data Types: RSA public key Key Number: 0 Length: 140

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
001C 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0046 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0062 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00
007E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 4.2.8-4 Key Pair for Remote Update

Select “Data Type” and get key value by directly editing hexadecimal value or importing from a file. Please see the following figure:



Look in: Local Disk (C:)

china-drm
Config.Msi
dell
Documents and Settings
dosh
Download
Downloads
FTJavaCard
Inetpub
Intel
localsymbols
MSOCache
Program Files
QUARANTINE
Rockey5
Ruby192
symbols
WINDOWS
workspace
wwwhost
Product.Pub

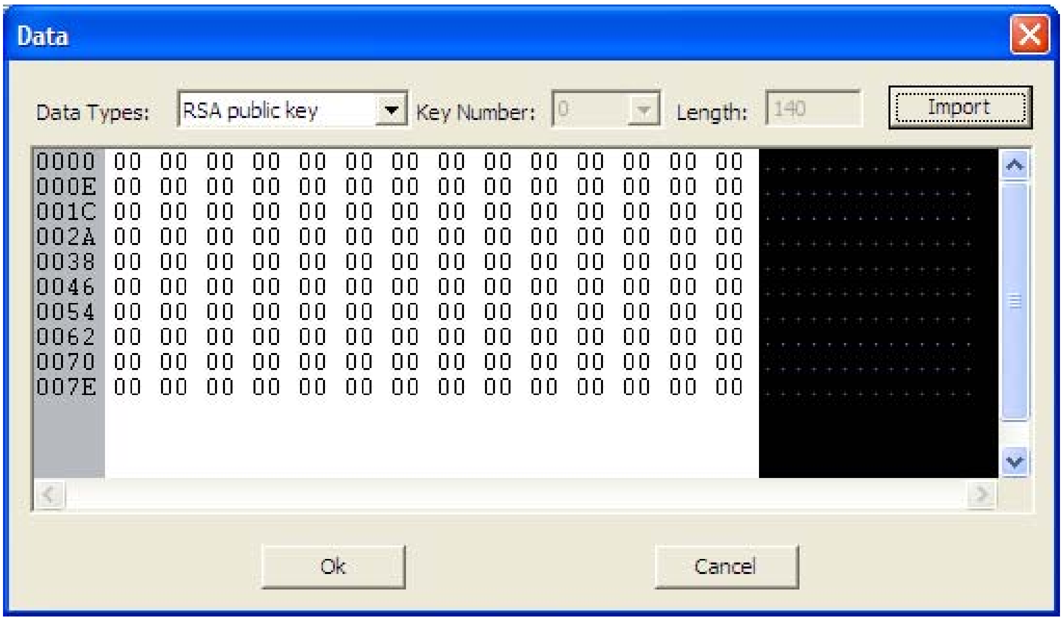
File name: Product.Pub

Files of type: Public Key Files (*.pub)

Figure 4.2.8-5 Select Key File

(4) RSA Key Pair 1

Double click “1 RSA public private ey Pair” to set key value. Please see the following figure:



Data Types: RSA public key
Key Number: 0
Length: 140

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
001C 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0046 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0054 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0062 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00
007E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 4.2.8-6 Set RSA Key Value

Select "Data Type" and get key value by directly editing hexadecimal value or importing from a file. Use the same way to set RSA Key Pair 2 to 8.

(5) User File System

Double click "20 User File System" to initialize Virtual File System. Please see following figure:

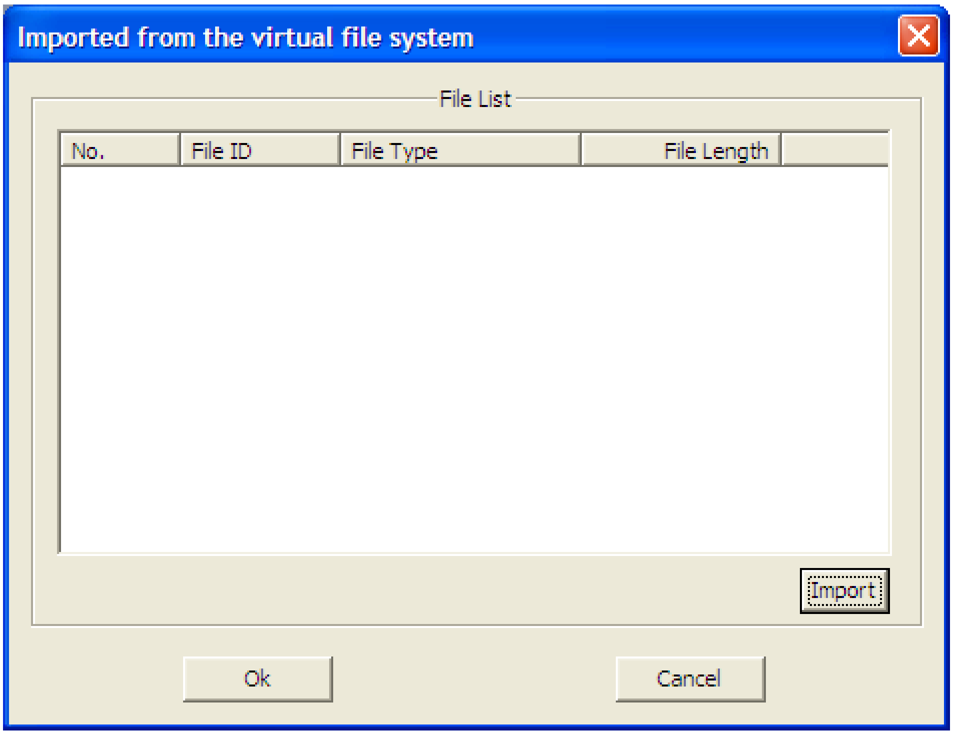


Figure 4.2.8-7 User File System

Click "import" to import a virtual file. A virtual file can be generated by VfsSet.exe (Virtual File System Manager).

When finished every item, click “Initialize” button to begin initialization. User Tool will notify success when finished. At this point, tick “Automatic Batch Initialization” to initialize ROCKEY5 TIME in batch. ROCKEY5 TIME LED is blinking when initializing. Change to a new ROCKEY5 TIME when LED is off.

4.3 ROCKEY5 TIME Virtual File System Manager (VFSM)

When software manufacturers develop C51 programs, KEIL and ROCKEY5 TIME Virtual File System Manger are needed. VFSM is a virtual environment that simulates real ROCKEY5 TIME. Customers do not need ROCKEY5 TIME when developing programs. VFSM can provide all ROCKEY5 TIME functions.

VFSM generates Virtual Files (.vfs), which can be used to create file system for real ROCKEY5 TIME.

VFSSet.exe is VFSM, which can be found under “Utilities” folder in DK.

4.3.1 Main Interface

ROCKEY5 TIME VFSM includes menu bar, tool bar, Tree View and information list. Please see the figure below:

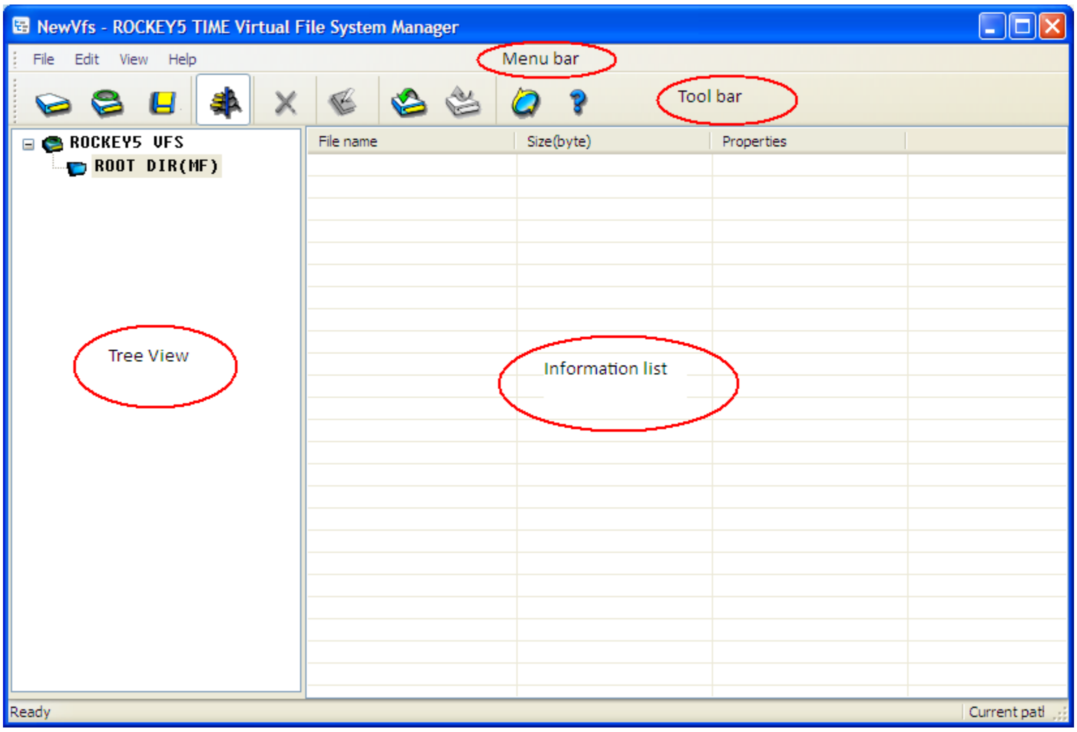


Figure 4.3.1-1 ROCKEY5 TIME VFSM

Menu bar: main functions are here.

Tool bar: Same as menu bar. Provides buttons for convenience.

Tree view: List folders of virtual file.

Information list: List detailed information of virtual file.

4.3.2 Create, Save and Open

(1) There are two ways to create a new virtual file.

First way: Click “File” in menu bar then click “New”. Please see figure below:

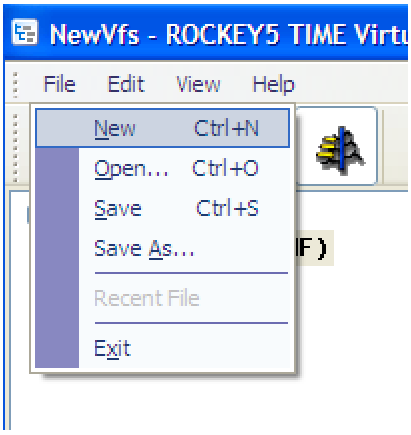


Figure 4.3.2-1 Create a New File

Second way: Click button on tool bar to create a new file.

(2) There are two ways to save files.

First way: Click “File” in menu bar and click “Save”. Please see figure below:

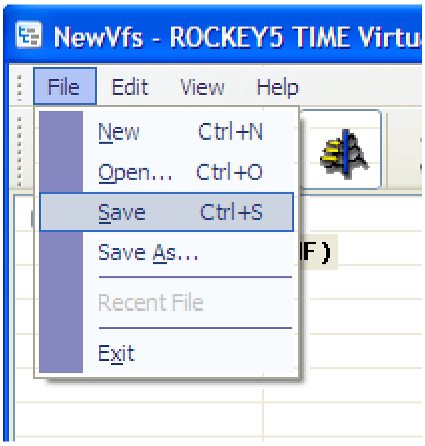
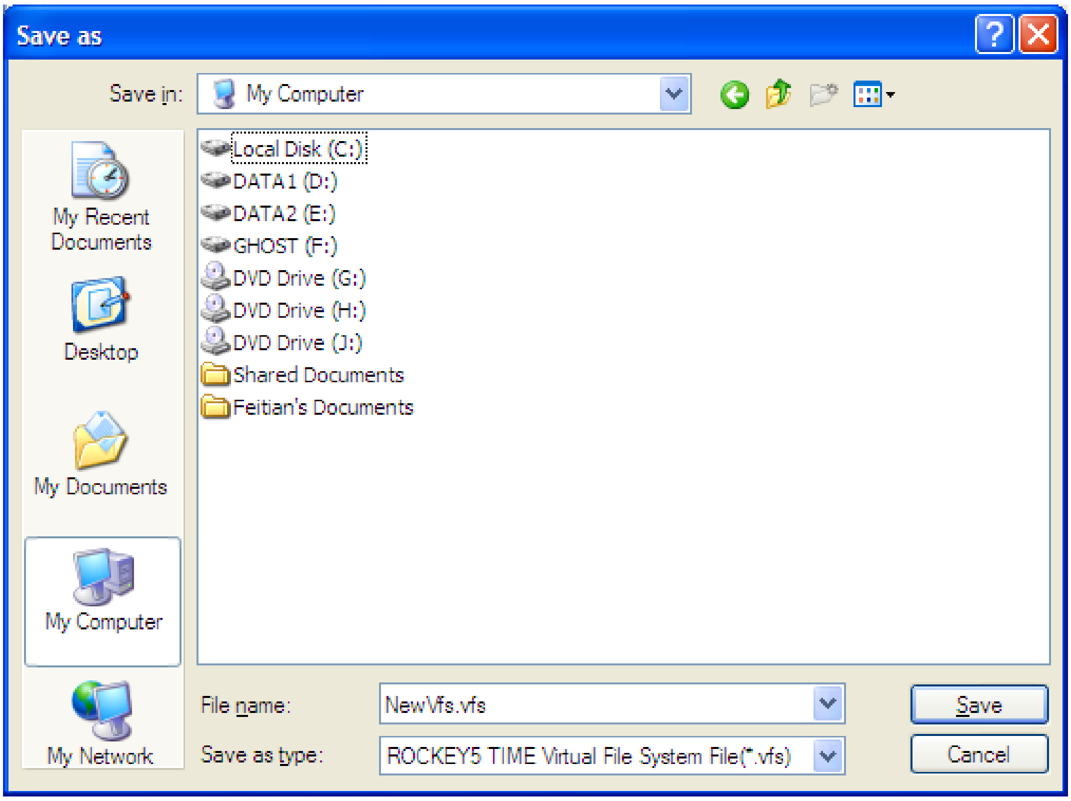


Figure 4.3.2-2 Save a File

Then select “save” and input correct path to save files into hard disk. Please see the figure below:



Save as

Save in: My Computer

Local Disk (C:)
DATA1 (D:)
DATA2 (E:)
GHOST (F:)
DVD Drive (G:)
DVD Drive (H:)
DVD Drive (J:)
Shared Documents
Feitian's Documents

File name: NewVfs.vfs

Save as type: ROCKEY5 TIME Virtual File System File(*.vfs)

Figure 4.3.2-3 Save a File Dialog

Second way: Click 💾 button on menu bar to save files.

(3) There are two ways to open an existed file.

First way: Click "File" in menu bar and then select "Open". Please see the figure below:

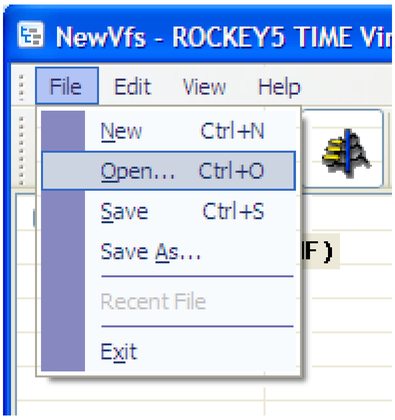
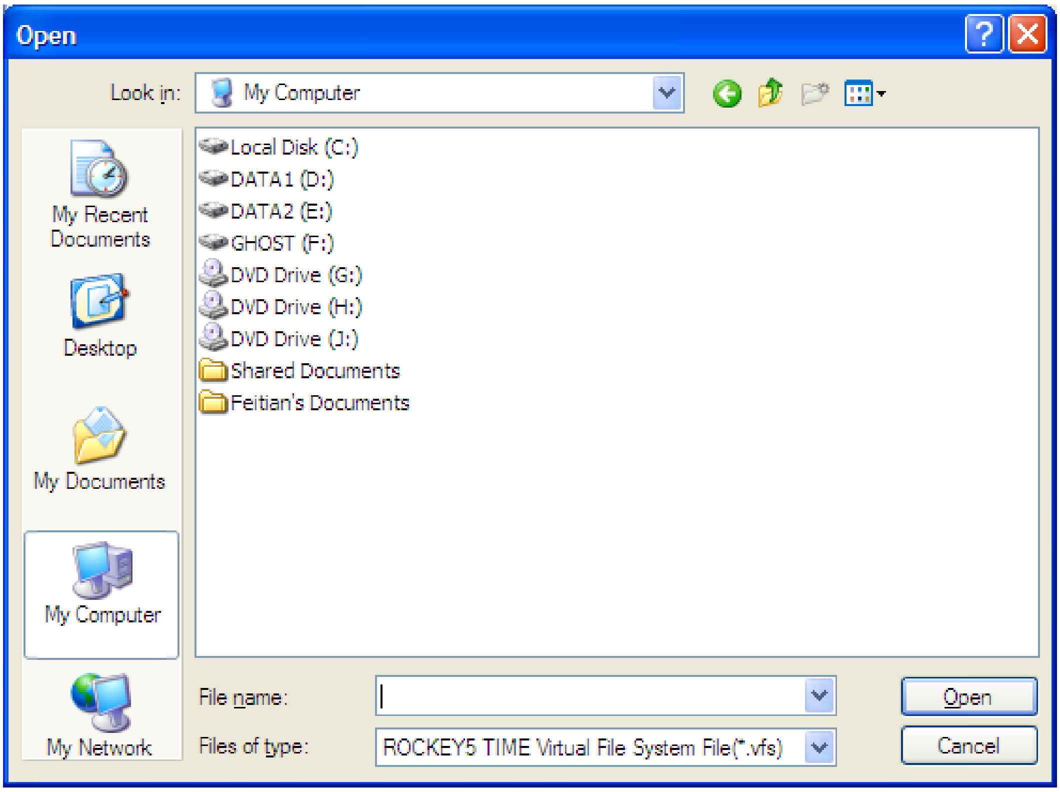


Figure 4.3.2-4 Open a File

Then input correct path in the pop-up dialog to open an existed file. Please see the figure below:



Look in: My Computer

Files of type: ROCKEY5 TIME Virtual File System File(*.vfs)

Figure 4.3.2-5 Open a File

Second way: Click button on tool bar to open an existed virtual file. Input correct path to open files.

4.3.3 Create a New Internal File

There are two ways to create a new internal file.

(1) Click “Edit” in menu bar and select “Create File”. Please see figure below:

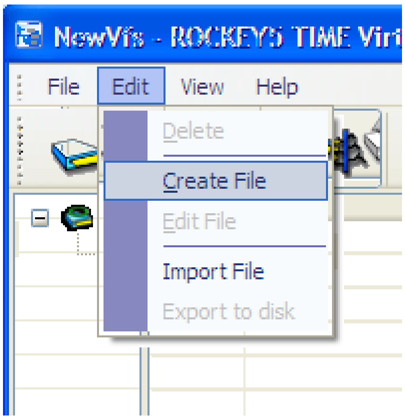
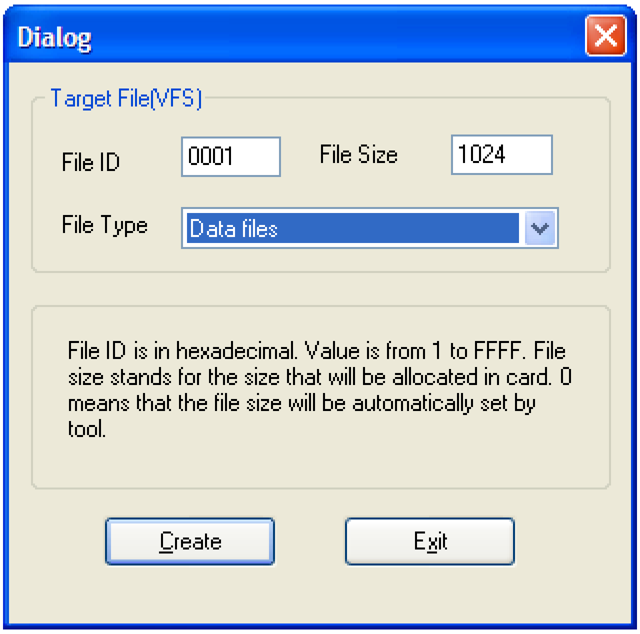


Figure 4.3.3-1 Create a File

Select “Create file” to show the following figure:

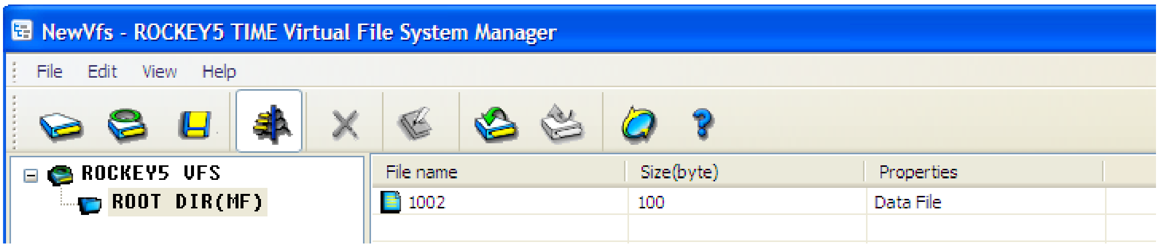


File ID: 0001
File Size: 1024
File Type: Data files

File ID is in hexadecimal. Value is from 1 to FFFF. File size stands for the size that will be allocated in card. 0 means that the file size will be automatically set by tool.

Figure 4.3.3.-2 Create a File

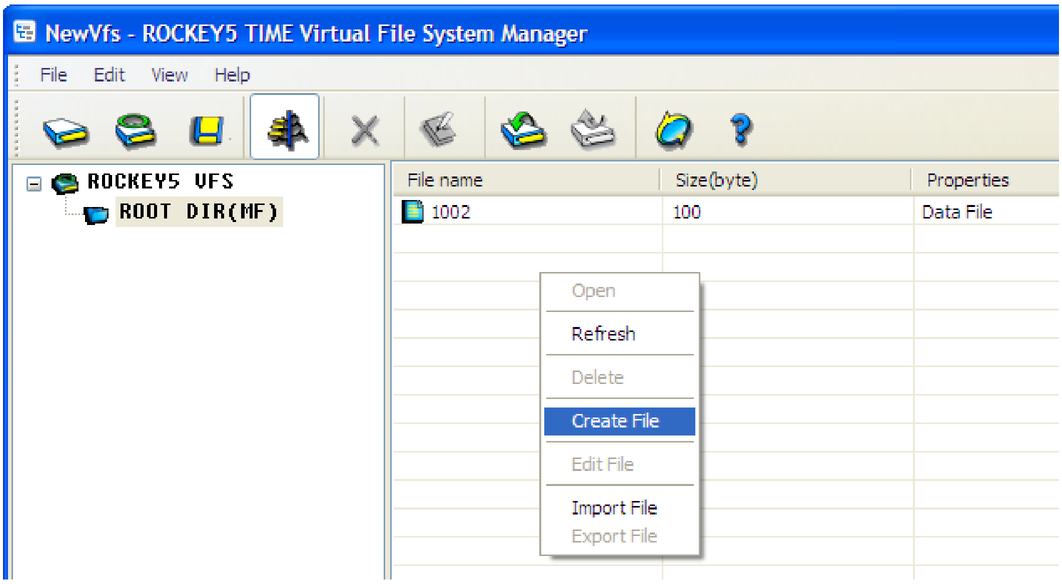
File ID, file size and file type are needed. Please note: File ID is 2 bytes long. Valid range is from 0x0001 to 0xFFFF. File size cannot exceed the remaining free space of ROCKEY5 TIME.



File name	Size(byte)	Properties
1002	100	Data File

Figure 4.3.3-3 After Created a File

(2) Right click on Information list and select “Create file” to create a new internal file.



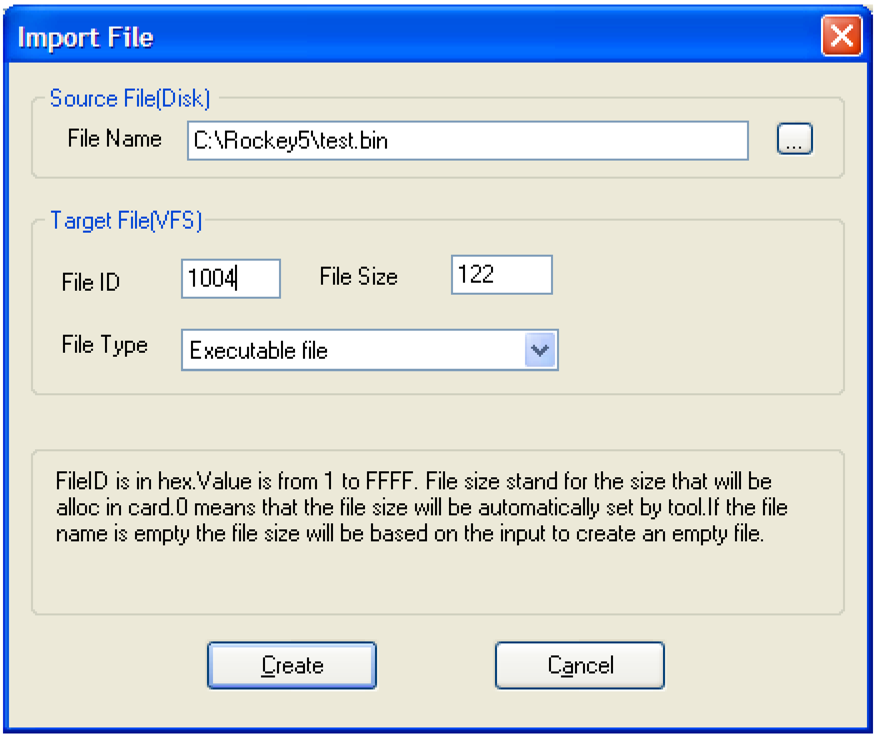
File name	Size(byte)	Properties
1002	100	Data File

Figure 4.3.3-4 Create a File

4.3.4 Import Files

Customers could import an existed file into virtual file system. There are two ways to import:

(1) Click button to import. Please see figure below:



File Name: C:\Rockey5\test.bin

File ID: 1004

File Size: 122

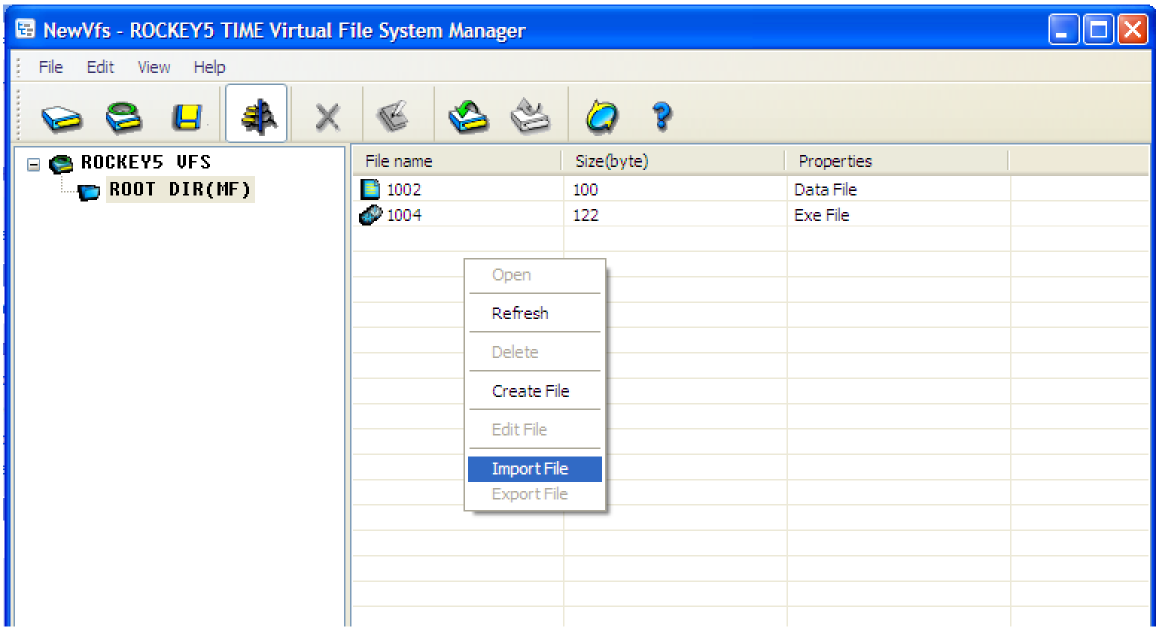
File Type: Executable file

FileID is in hex.Value is from 1 to FFFF. File size stand for the size that will be alloc in card.0 means that the file size will be automatically set by tool.If the file name is empty the file size will be based on the input to create an empty file.

Figure 4.3.4-1 Import a File

Input a file in "Source File (Disk)" field. File size will be automatically filled in. When manually input a file size, if input file size is bigger than actual file size, file will be padded with 0. If input file size is smaller than actual file size, file will be cut.

(2) Right click on information list and select "Import File" to import a file. Please see the figure below:



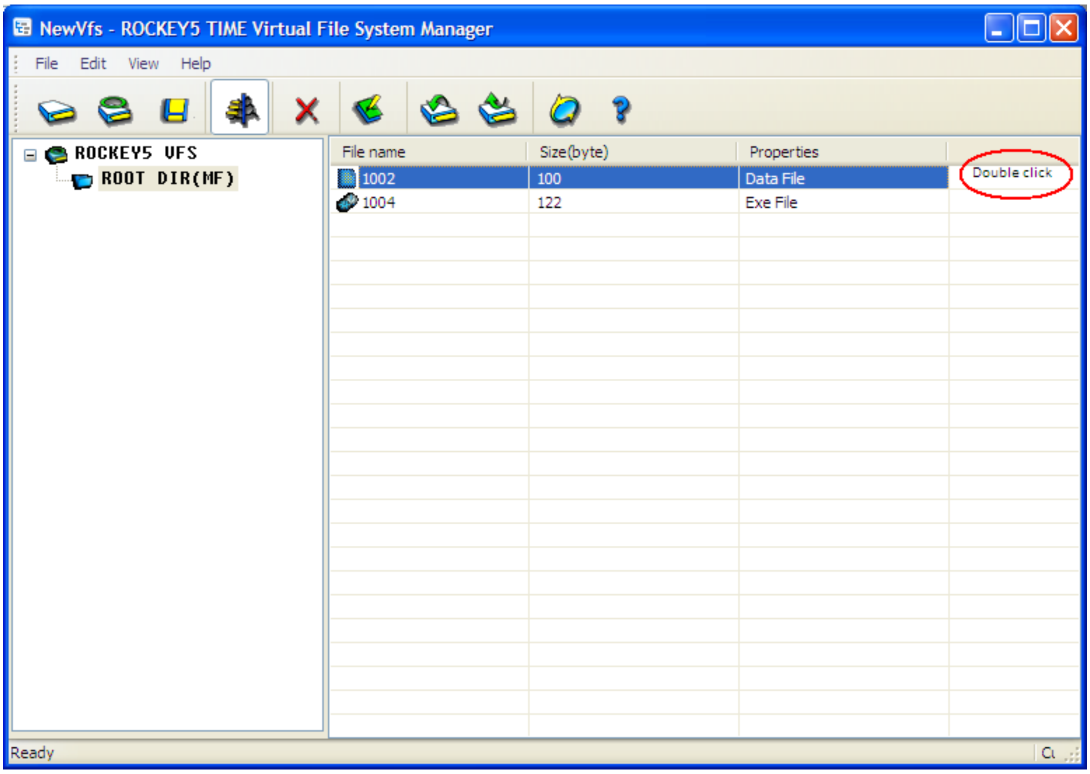
File name	Size(byte)	Properties
1002	100	Data File
1004	122	Exe File

Figure 4.3.4-2 Import a File

4.3.5 Open an Internal File

There are two ways to open an internal file.

(1) Double click files in information list to open files. Please see the figure below:

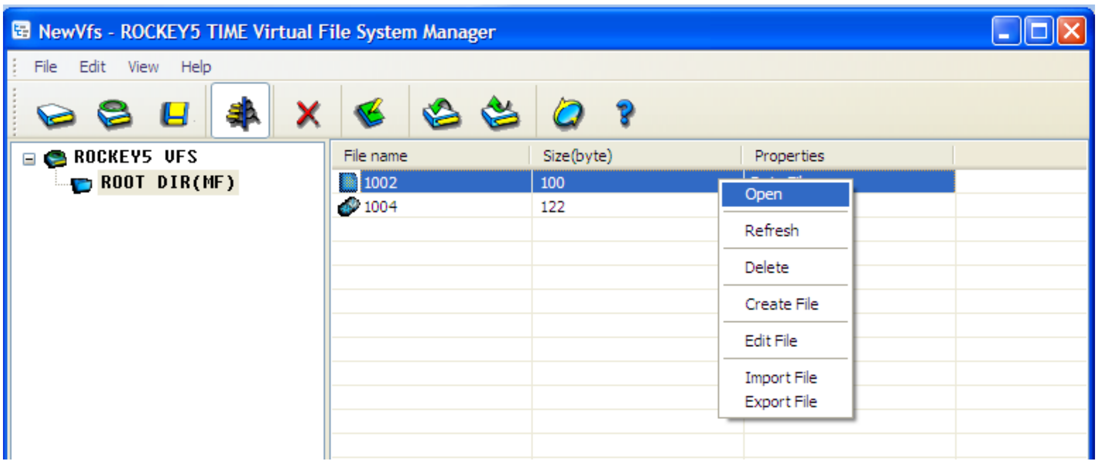


File name	Size(byte)	Properties
1002	100	Data File
1004	122	Exe File

Figure 4.3.5-1 Double Click to Open

Open file to edit. Later we will introduce how to edit a file.

(2) Right click files in information list and select “Open” to open a file. Please see the figure below:



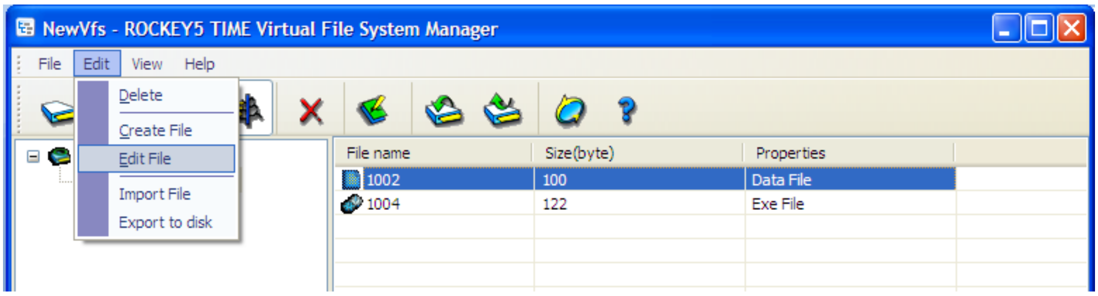
File name	Size(byte)	Properties
1002	100	
1004	122	

Figure 4.3.5-2 Open a File

4.3.6 Edit a File

In virtual file system, customers can edit any files. There are three ways to edit files.

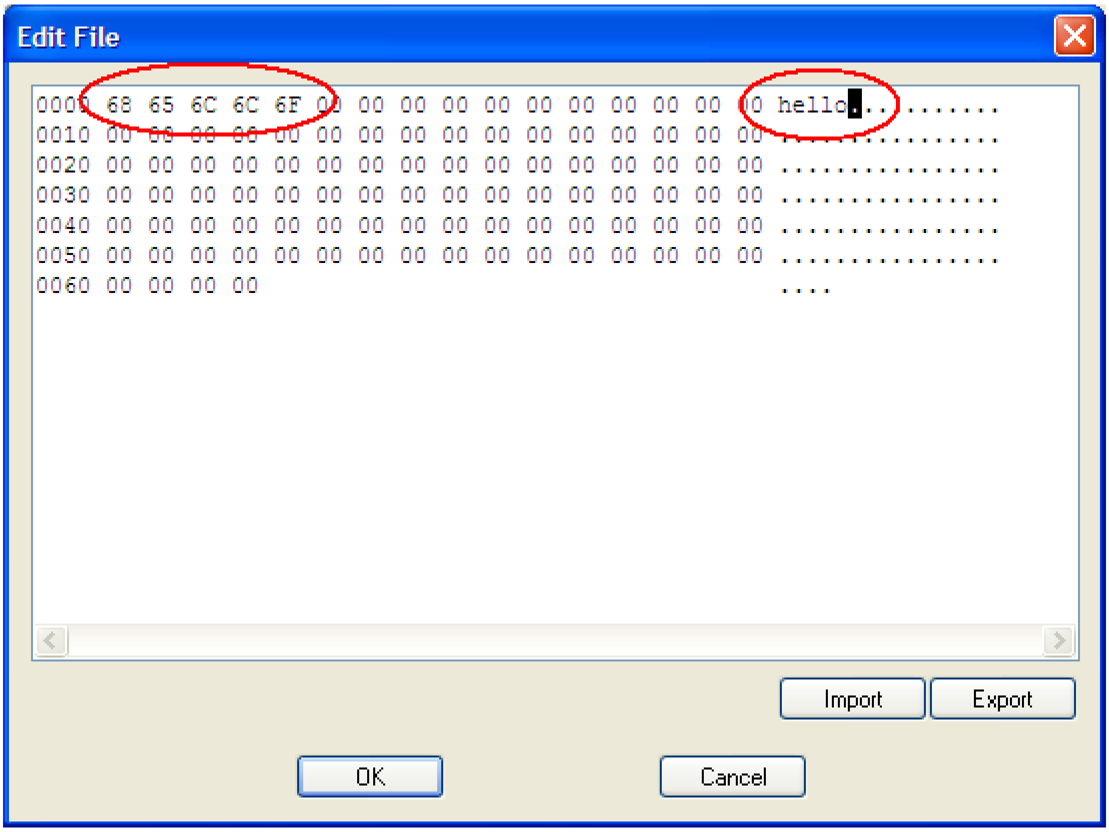
(1) Select a file and then go to menu bar, select “Edit file”. Please see figure below:



File name	Size(byte)	Properties
1002	100	Data File
1004	122	Exe File

Figure 4.3.6-1 Edit File

Here is the Edit interface:



```
0000 68 65 6C 6C 6F 00 00 00 00 00 00 00 00 00 00 00 hello...........
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060 00 00 00 00                                     ....
```

Figure 4.3.6-2 Edit Interface

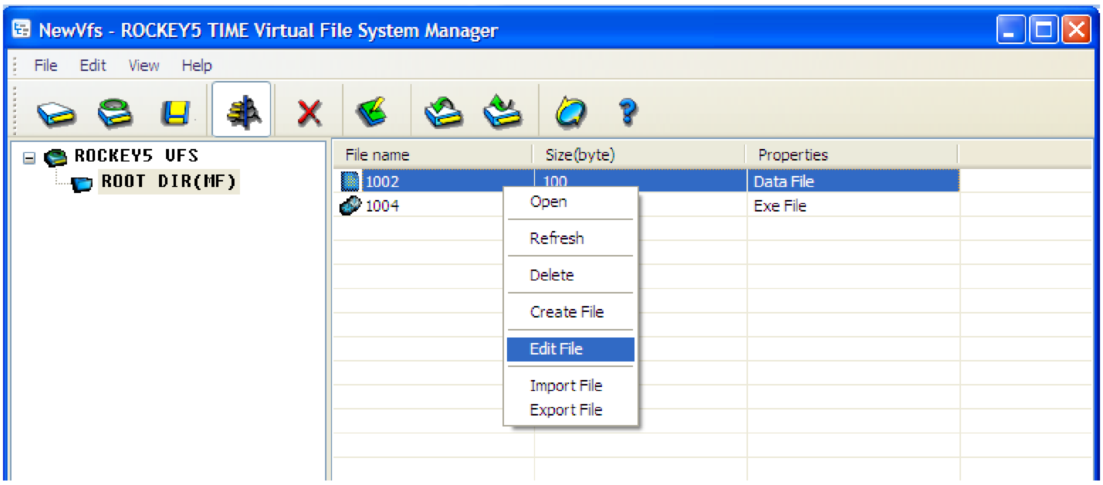
Customers can directly edit the hexadecimal data or ASCII data.

“Import” button allows customers to import a file into virtual file system. If imported file is bigger than current file, imported file will be cut. If imported file is smaller than the current file, remaining part will be kept.

”Export” button will export the current file to hard disk.

(2) Select a file and then click button to edit.

(3) Right click a file then select “Edit file” to edit. Please see the figure below:



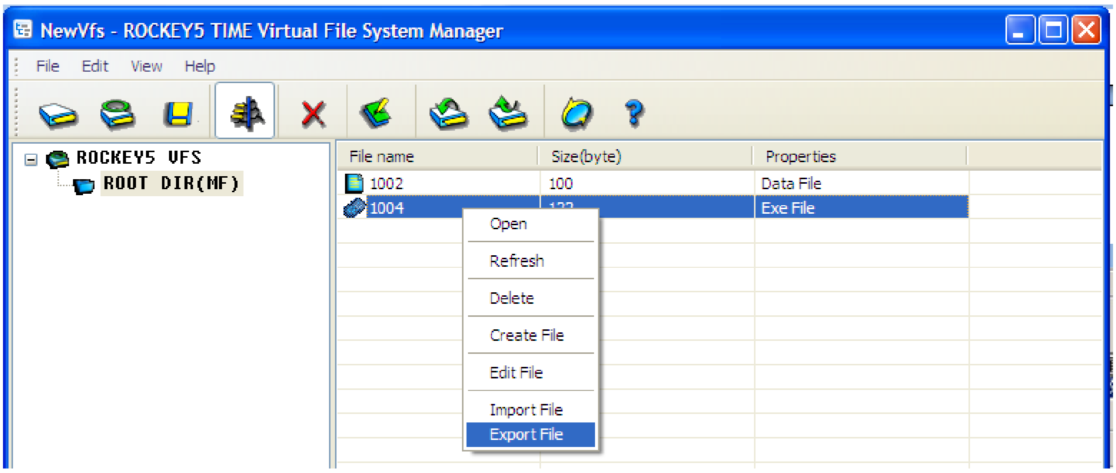
File name	Size(byte)	Properties
1002	100	Data File
1004		Exe File

Figure 4.3.6-3 Edit a File

4.3.7 Export a File

There are three ways to export an internal file.

(1) Right click a file in information list and select "Export File" to export a file.

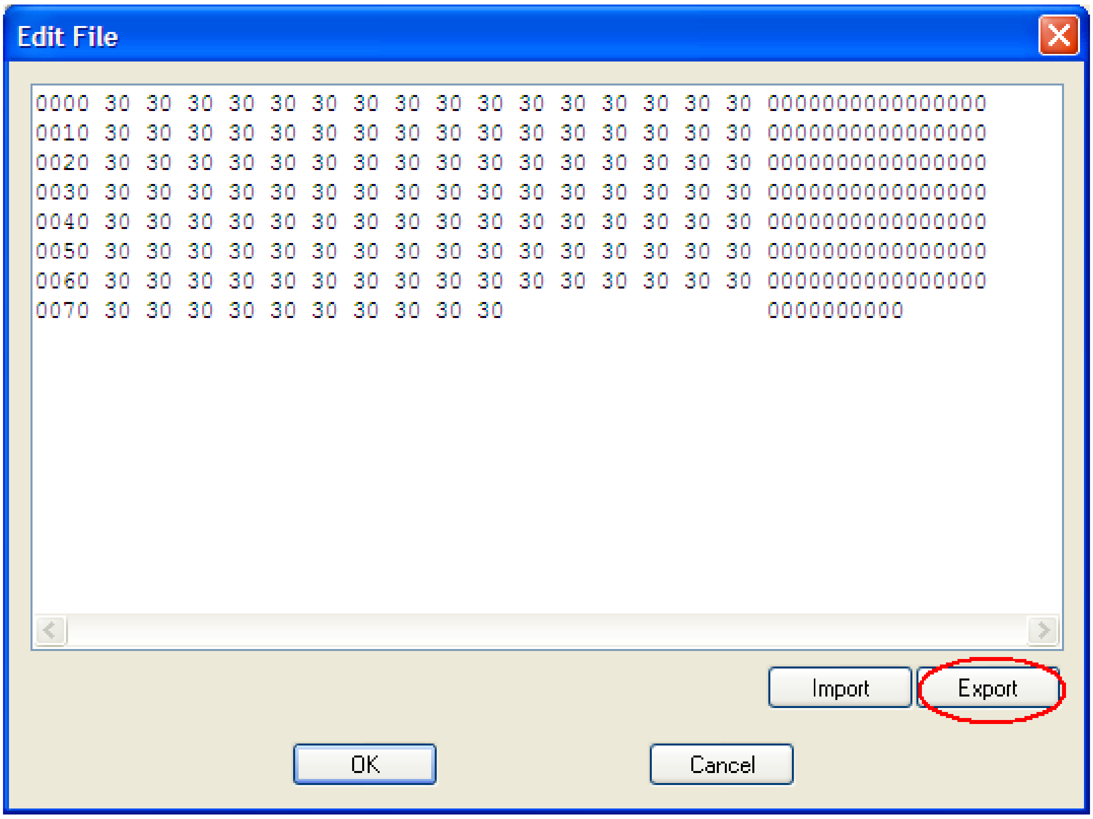


File name	Size(byte)	Properties
1002	100	Data File
1004	122	Exe File

Figure 4.3.7-1 Export a File

(2) Click button to export a file.

(3) Export data into a file in edit interface. Please see figure below:



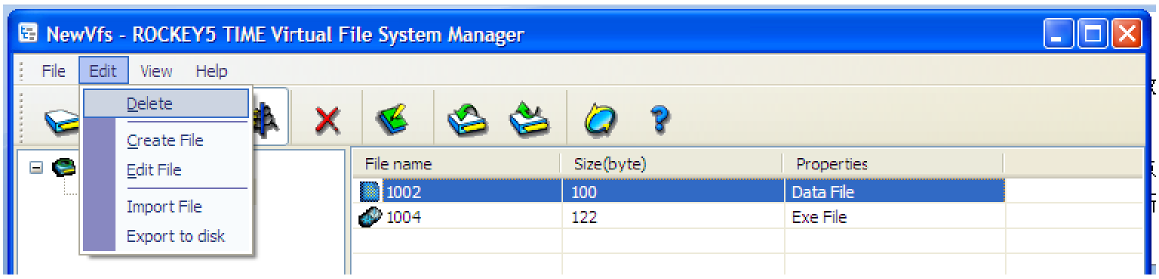
```
0000  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  0000000000000000
0010  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  0000000000000000
0020  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  0000000000000000
0030  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  0000000000000000
0040  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  0000000000000000
0050  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  0000000000000000
0060  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  0000000000000000
0070  30  30  30  30  30  30  30  30  30  30                          0000000000
```

Figure 4.3.7-2 Export to File

4.3.8 Delete a File

There are three ways to delete files in virtual file system:

(1) Click “Edit” in menu bar and select “Delete File”. Please see the figure below:

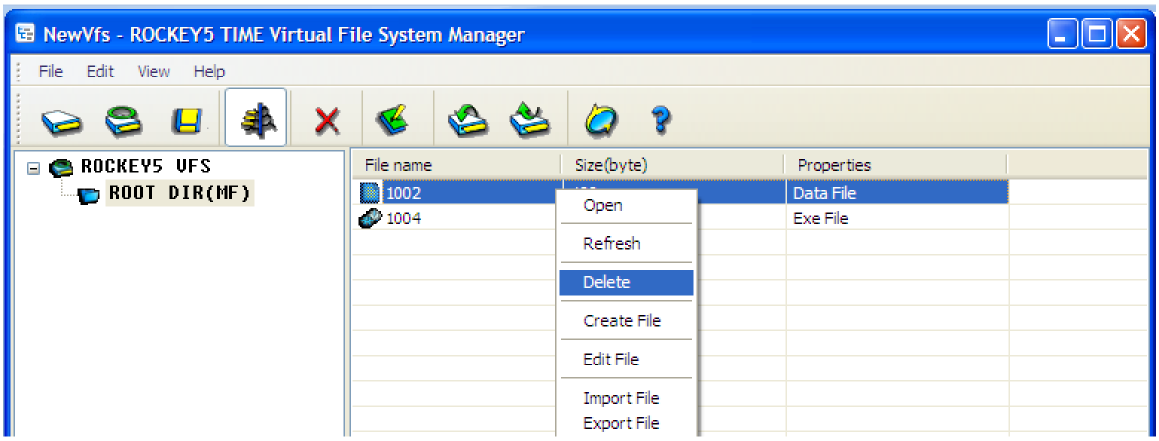


File name	Size(byte)	Properties
1002	100	Data File
1004	122	Exe File

Figure 4.3.8-1 Delete a File

(2) Select a file and then click ✕ button on tool bar to delete.

(3) Right click a file and select “Delete” to delete a file. Please see the figure below:



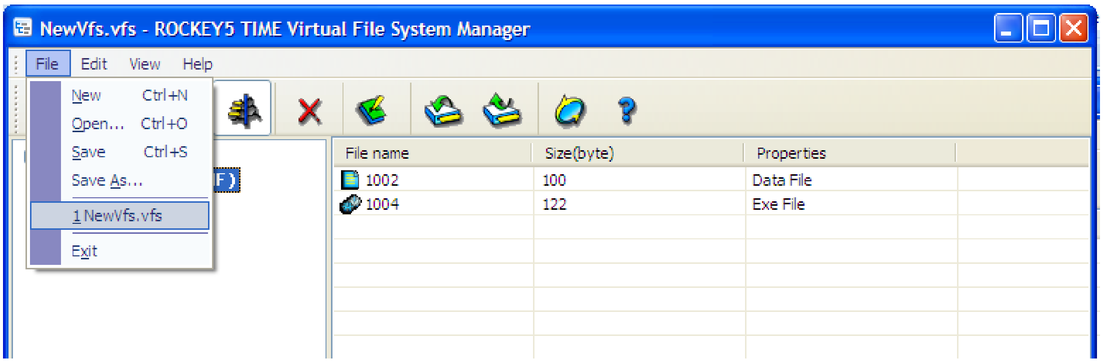
File name	Size(byte)	Properties
1002		Data File
1004		Exe File

Figure 4.3.8-2 Delete a File

4.3.9 Other Functions

(1) Open recent virtual files.

Click “File” in menu bar, customers will find recent files listing on drop down menu. Please see the figure below:

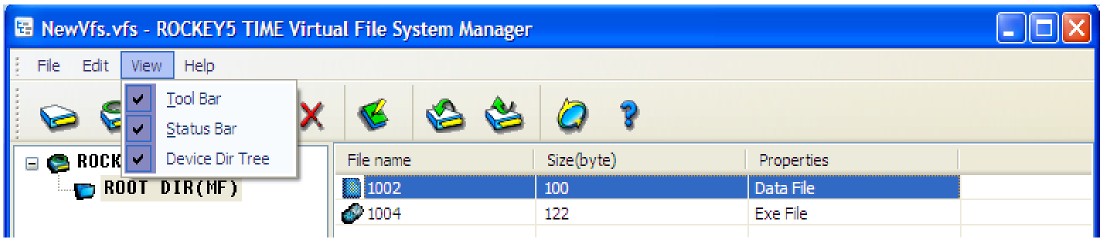


File name	Size(byte)	Properties
1002	100	Data File
1004	122	Exe File

Figure 4.3.9-1 Open Recent Files

(2) Hide Tree View. Click button to hide Tree View.

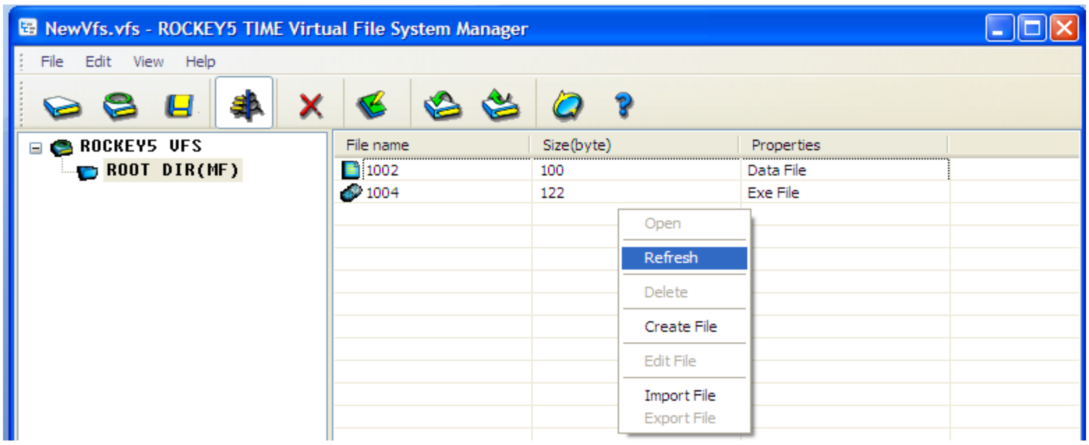
(3) View Menu: To show or hide tool bar, Tree View and status bar.



File name	Size(byte)	Properties
1002	100	Data File
1004	122	Exe File

Figure 4.3.9-2 View

(4)Refresh: Sometimes, when files or directories are changed, customers need to refresh the tool. Click button to refresh. Please see the figure below:



File name	Size(byte)	Properties
1002	100	Data File
1004	122	Exe File

Figure 4.3.9-3 Refresh

4.3.10 Menu Bar and Tool Bar

New	Create a new virtual file
Open	Open an existed virtual file
Save	Save virtual file in default type
Save as	Save virtual file in other type
Exit	Exit VFSM

Table 4.3.10-1 Menu

Delete	Delete a file
New File	Create a file
Edit File	Edit a file
Import File	Import a file
Export File	Export a file

Table 4.3.10-2 Edit

Tool bar	Hide/Show tool bar
Status bar	Hide/Show status bar
Tree View	Hide/Show Tree View

Table 4.3.10-3 View

	New
	Open
	Save
	Hide/Show Tree View
✗	Delete
	Edit
	Import
	Export
	Refresh
?	About

Table 4.3.10-4 Tool Bar

4.4 ROCKEY5 TIME Envelope

ROCKEY5 TIME provides Envelope to protect software. Envelope enables to protect software without writing any code. Even a non-technical person, he or she can protect software in several minutes. Envelope provides a flexible and strong protection.

ROCKEY5 TIME Envelope supports lots of encryption techniques:

■ Compressing, encrypting and re-building PE files

Envelope uses high performance compression algorithm to compress code segment, data segment, import table, resource segment and Envelope itself. Compressed programs are smaller and hard to be reversed.

■ MD5 to ensure the integrity of protected files

Envelope uses MD5 algorithm to get a hash code. This hash code is called the fingerprint of the file. If protected files have been modified, the fingerprint will be changed. This way is used to avoid programs being modified.

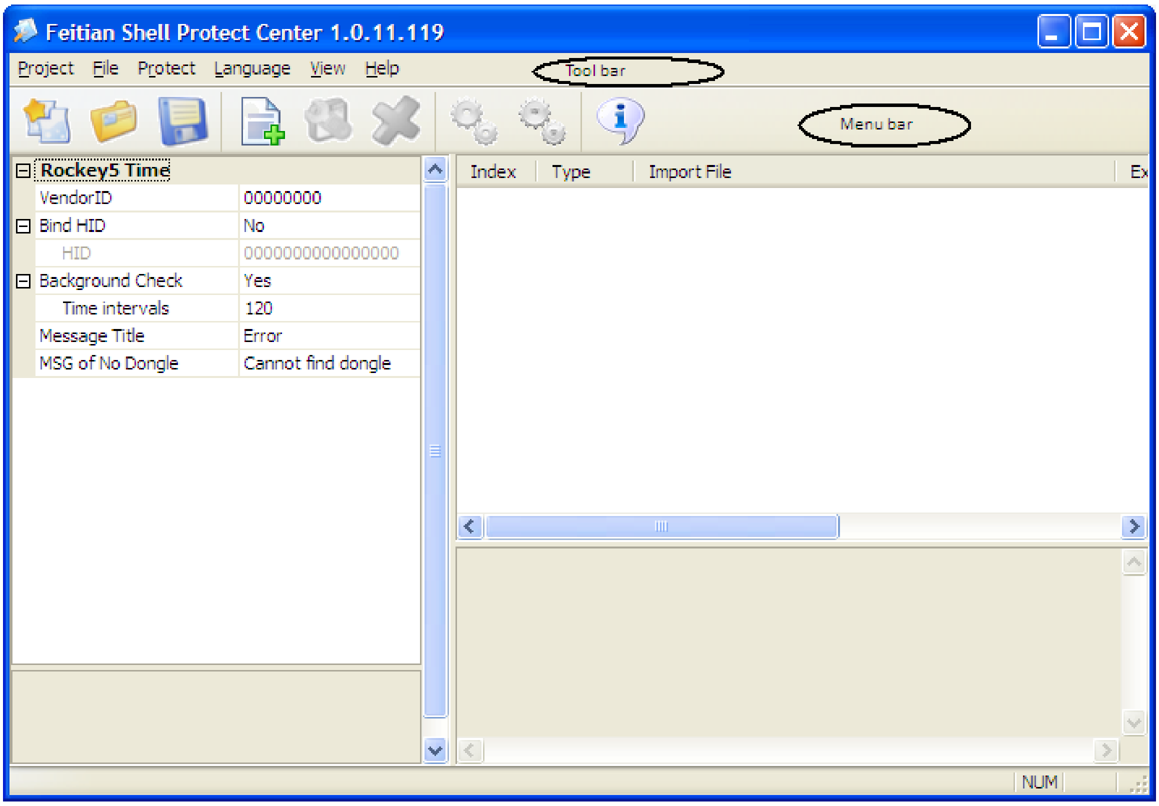
■ Anti-debug and anti-trace

Envelope uses lots of anti-trace and anti-debug methods. Such as parent process checking, memory checking and forbidding debug instructions.

■ Memory checking

Envelope creates a monitor thread to check memory. If the read-only code segment or Envelope code has been modified, the protected program will be terminated.

Envelope is under “Utilities\Envelope” folder in DK. This tool needs RockeyCmdShell.exe. It is recommended to protect software with both Envelope and API. The Envelope is Envelope.exe. Here is the figure of Envelope:


Feitian Shell Protect Center 1.0.11.119
Project File Protect Language View Help
Tool bar
Menu bar
Index Type Import File Ex
NUM

Rockey5 Time	
VendorID	00000000
Bind HID	No
HID	0000000000000000
Background Check	Yes
Time intervals	120
Message Title	Error
MSG of No Dongle	Cannot find dongle

Figure 4.4-1 Envelope

Select languages in menu bar to change language interface:

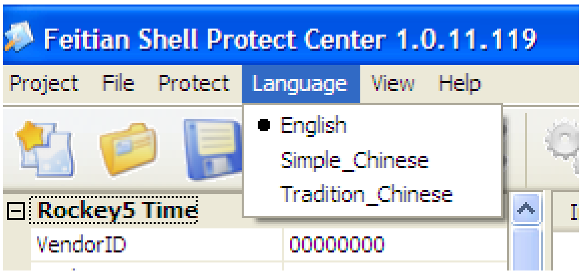
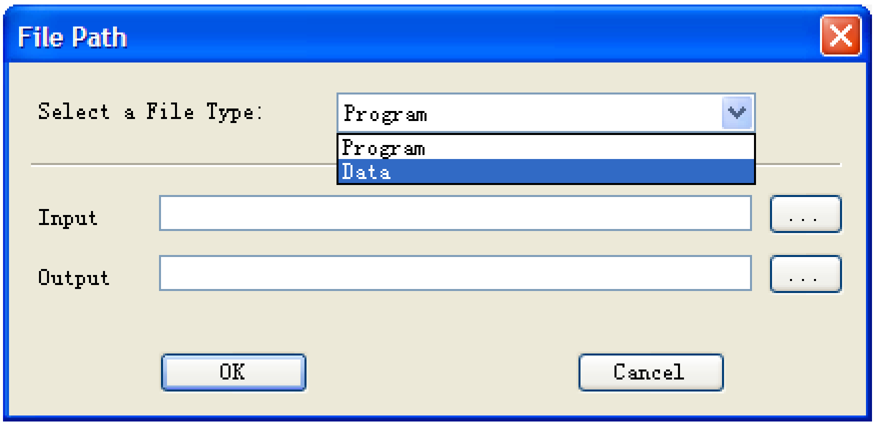

Feitian Shell Protect Center 1.0.11.119
Project File Protect Language View Help
● English
Simple_Chinese
Tradition_Chinese
Rockey5 Time
VendorID
00000000

Figure 4.4-2 Language

Click File->Add file or click button to add files. In the adding file dialog, select a file type. Please see the figure below:



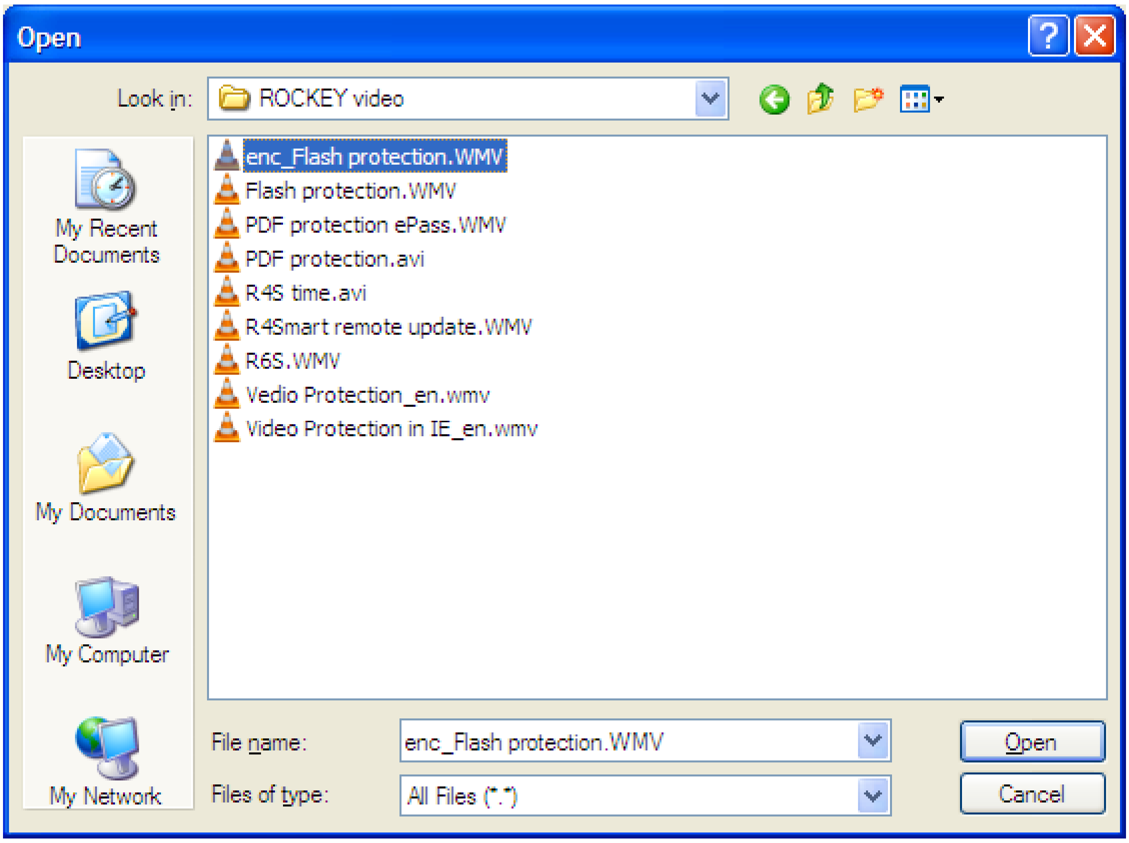
Select a File Type: Program

Program
Data

Figure 4.4-3 Select a File

File Type: There are two file types: Program and Data. If customers want to protect programs, please add PE files or .Net files, such as EXE, DLL, ARX, VB.Net and C#.Net. If select data, please add PDF, Flash and Video files.

1> If select Data, click ... button to add files. Please see the figure below:



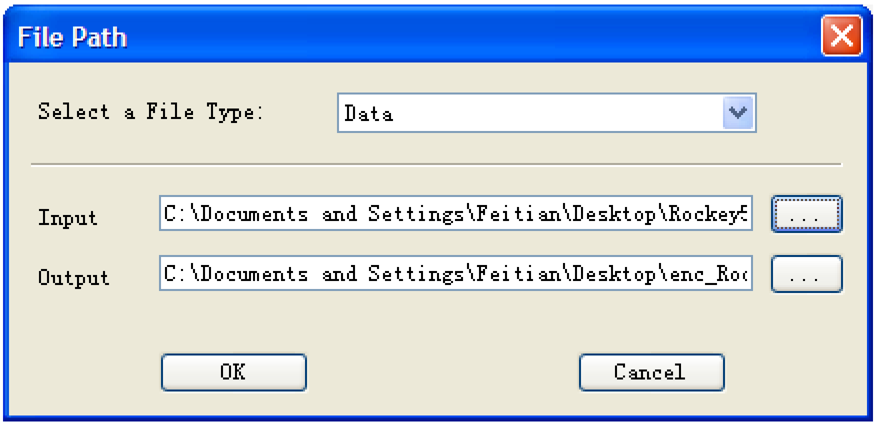
Look in: ROCKEY video

enc_Flash protection.WMV
Flash protection.WMV
PDF protection ePass.WMV
PDF protection.avi
R4S time.avi
R4Smart remote update.WMV
R6S.WMV
Vedio Protection_en.wmv
Video Protection in IE_en.wmv

File name: enc_Flash protection.WMV
Files of type: All Files (*.*)

Figure 4.4-4 Add Files

After selected, Envelope will automatically generate output path. Please see this figure:



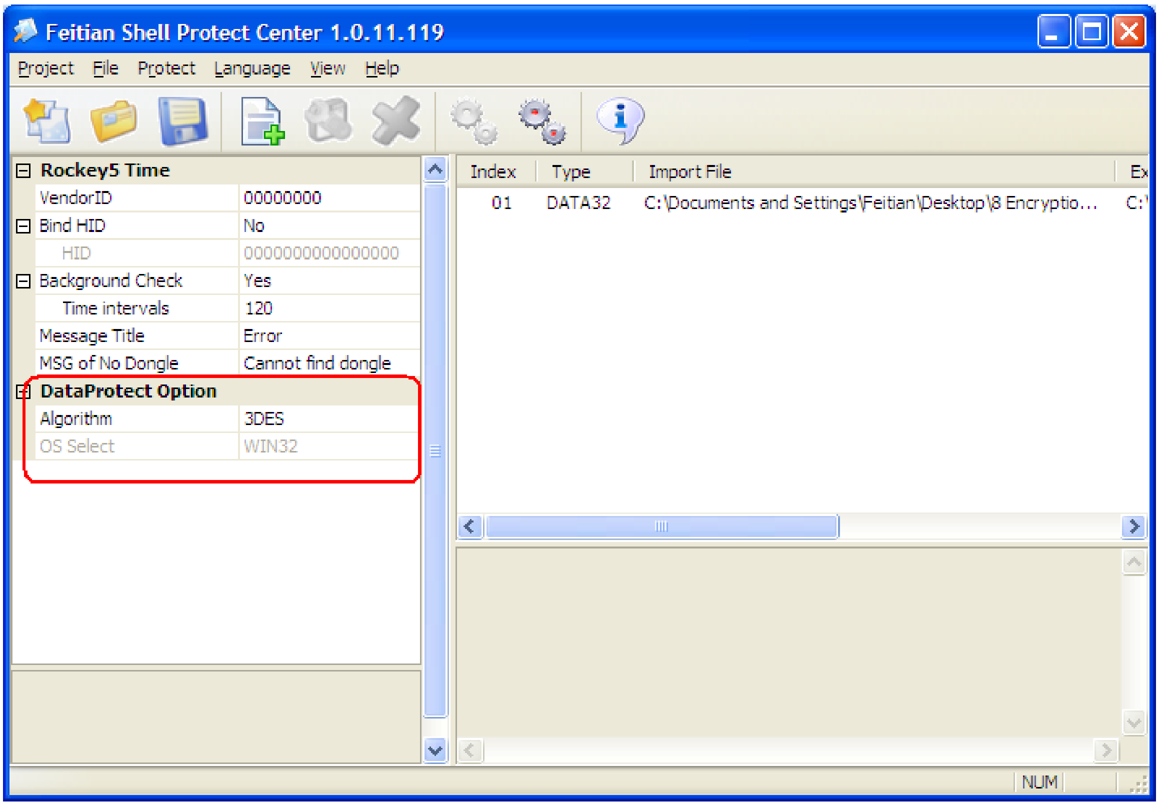
Select a File Type: Data

Input C:\Documents and Settings\Feitian\Desktop\Rockey5

Output C:\Documents and Settings\Feitian\Desktop\enc_Roc

Figure 4.4-5 File Path

Click “OK” to continue and you’ll see the figure below:



Feitian Shell Protect Center 1.0.11.119
Project File Protect Language View Help
NUM

Rockey5 Time	
VendorID	00000000
Bind HID	No
HID	0000000000000000
Background Check	Yes
Time intervals	120
Message Title	Error
MSG of No Dongle	Cannot find dongle
DataProtect Option	
Algorithm	3DES
OS Select	WIN32

Index	Type	Import File	Ex
01	DATA32	C:\Documents and Settings\Feitian\Desktop\8 Encryptio...	C:\

Figure 4.3-6 Envelope

In left panel, there are several options:

■Vendor ID

Vendor ID is mandatory to be input. The default Vendor ID is “00000000”. Please change it before using. Please refer section2.2.

■Bind HID

If selected “Bind HID”, the protected programs can be run only if the specified HID ROCKEY5 TIME is attached.

■ Background Check

If selected “Background Check”, the protected programs will be checked every period of time. The time intervals should be greater than 120 seconds.

■ Message Title

Input the title of ROCKEY5 TIME error message.

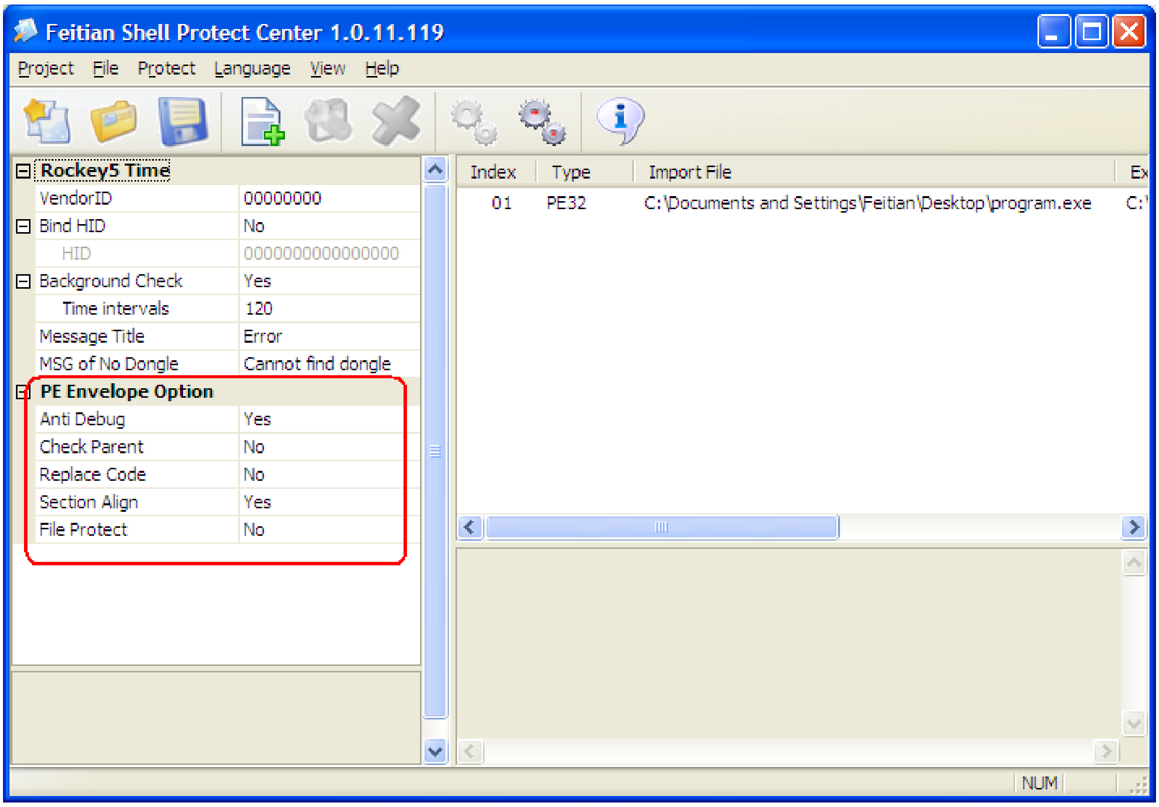
■ Error Message

Input the error message of ROCKEY5 TIME.

■ DataProtect Options

There are two options. The first one is to select encryption algorithm (3DES or RC4). The other one is to select OS.

1> If selected “Program”, follow Figure 4.3-3, Figure 4.3-4 and Figure 4.3-5 to add files. Please see the figure below:


Feitian Shell Protect Center 1.0.11.119
Project File Protect Language View Help
NUM

⊟ Rockey5 Time	
VendorID	00000000
⊟ Bind HID	No
HID	0000000000000000
⊟ Background Check	Yes
Time intervals	120
Message Title	Error
MSG of No Dongle	Cannot find dongle
⊟ PE Envelope Option	
Anti Debug	Yes
Check Parent	No
Replace Code	No
Section Align	Yes
File Protect	No

Index	Type	Import File	Ex
01	PE32	C:\Documents and Settings\Feitian\Desktop\program.exe	C:\

Figure 4.3-7 Envelope

Please note: For program protection, there are several PE/.Net options. Here will only introduce PE/.Net options. Others are same as Figure 4.3-6.

■ Anti Debug

Add some techniques to avoid being debugged.

■ Check Parent

Protected programs will detect debugger process. If there is a debugger process, protected programs cannot be run.

■ Replace Code

Replace some code of programs.

■ Section Align

Align the program sections.

■ File Protect

Check the integrity of the protected file to avoid files being modified.

Click ⚙ button to encrypt a single file. Click ⚙ button to encrypt all files.

If error occurred, please follow below steps:

(1) Check whether Vendor ID is correctly input.

(2) Check whether the file type is supported. Please use C:\WINDOWS\NOTEPAD.EXE to test.

Chapter 5. Advanced Use of ROCKEY5 TIME

Previous chapter described how to use ROCKEY5 TIME to protect software. In this chapter, we will give a sample to show the advanced use of ROCKEY5 TIME. The sample is a program that updates internal data from 15 bytes long to 18 bytes long.

```
#include "stdafx.h"
#include <stdio.h>
#include <windows.h>
#include "RY5_API.h"

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

void main(int argc, char* argv[])
{
    char cOldID[16] = "110105720924001";//15 bytes long data
    char cNewID[19] = {0};              // 18 bytes long data
    ConvertID(cOldID, cNewID);          // go to convert
    printf("%s\n", cNewID);
}
```

Here we re-write the conversion algorithm in a C51 manner：

```
#include "RY5_C51.h"
#include <string.h>

unsigned char Wi[18] = {7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2,1};
char Ai[11]={'1','0','x','9','8','7','6','5','4','3','2'};
void main(void)
{
    int i,j,s;
    byte ID[15], newID[18];

    s=0;
```

```
        if(wInLen != 15)//Whether the length of input data is 15 bytes long.
            _exit();

        memcpy(ID, pbInBuff, 15); // Store input data into an array.
        memcpy(newID,ID,6);
        newID[6]='1';
        newID[7]='9';
        memcpy(newID+8,ID+6,9);
        for(i=0;i<17;i++)
        {
            j=(newID[i]-48)*Wi[i];
            s+=j;
        }
        s%=11;
        newID[17]=Ai[s];
        _set_response(18, newID); // Output new data
        _exit();                             //Exit
    }
```

In C51 program, there are two parameters—wInLen and pbInBuff. wInLen stores the length of the data which is passed by external programs. pbInBuff stores input data. Please refer section 6.2.1. Customers can use KEIL to compile C51 programs and build binary files. Binary files can be burned into ROCKEY5 TIME by using the following two methods：

(1) Use ROCKEY5 TIME utilities. Please refer section 4.2.。

(2) Use ROCKEY5 TIME API. Please refer section 7.1.16 and 7.1.17.。

In this example, we imported the binary into ROCKEY5 TIME and set the File ID to 0x0001. Here is the modified external program：

```
    #include "stdafx.h"
    #include <stdio.h>
    #include <windows.h>
    #include "RY5_API.h"

    void main(int argc, char* argv[])
    {
        BYTE cOldID[16] = "110105720924001";
        BYTE cNewID[20] = {0};
        int newLen=19;
        char vendorID[10] = "A8C3E6FD";

        RY_HANDLE handle;
        DWORD dwRet = 0;
        int dwCount=0;


    //Find attached ROCKEY5 TIME and returns the number of attached ROCKEY5 TIME.
        dwRet=RY5_Find(vendorID,&dwCount);
        if(dwRet != RY5_SUCCESS && dwRet)
        {
            printf("RY5_Find Error:0x%08x\n", dwRet);
            return;
        }


        //Open ROCKEY5 TIME
        dwRet = RY5_Open(&handle,1);
        if(dwRet != RY5_SUCCESS)
        {
            printf("RY5_Open Error:0x%08x\n", dwRet);
            return;
        }


        //Execute internal executable file. File ID is 0x0001.
        dwRet = RY5_ExecuteFile(handle, 0x0001, cOldID, 15, cNewID, &newLen);
        if(dwRet != RY5_SUCCESS)
        {
            printf("RY5_ExecuteFile Error:0x%08x\n", dwRet);
            return;
        }

        //Close ROCKEY5 TIME
        RY5_Close(handle,true);


        //Print out the result
        printf("%s\n", cNewID);
    }
```

From this example, it is obviously that we cannot find any clues about the conversion algorithm. All algorithm entity is inside ROCKEY5 TIME. Crackers trying to crack on PC will only get old data and new data. They cannot reverse the algorithm. And the new data will be got only when ROCKEY5 TIME is attached. Due to the highest security level smart card chip, the internal programs cannot be obtained by any person, even the dongle manufacturers. The ROCKEY5 TIME cannot be cracked if internal programs are complex enough and internal programs have not been leaked out.

Chapter 6. ROCKEY5 TIME System Function (C51)

The features have been described above including tool demonstration and encryption process. In the following section, the ROCKEY5 TIME system function interface is introduced. The program on ROCKEY5 TIME is based on C51, which is basically identical to C. As a subset of C, it is specialized on hardware development with rich libraries.

There are three different modes when using C51 library: compact mode (compact_mode.LIB), small mode (small_mode.LIB) and large mode (large_mode.LIB). Small mode or large mode is highly recommended. Due to the limit addressing range, compact mode needs to be carefully used. Note that to set different libraries, just modify "Memory Model"(check 4.1.2 for more detail).

The memory is divided into three areas: 128 bytes data area, 128 bytes idata area and 2k+256bytes xdata area. The data area and xdata area stores C51's variable and parameter. The idata area is for exchanging argument in register, if program declaration in this area, it may be lost.

- **Small mode (small_mode.LIB)**: default variable declaration is in data area, except adding xdata when declaration, it will be in xdata area.
- **Large mode (large_mode.LIB)**: default variable declaration is in xdata area.

When declaring the variable, xdata need to add (for example int xdata i=0;). Without xdata, in small mode, the declared variables are in 128 bytes data area. For the array over 128 bytes, with xdata added, the variable will be allocated to 2k xdata area.

When assigning a constant, you'd better add code symbol (for example: const int code I = 8) to be allocated to code area. In this way, it only takes executable area rather than memory area.

Some tips to enhance the operating speed in dongle:

- Transfer some operations that efficiency cost but not related to security to the computer side to run, for instance: _swap.
- Transfer jump instruction to computer side. (For example: swith-case)

6.1 Exit

void _exit()

Objective:

To exit the C51 program running in ROCKEY5 TIME. The program will be exited and stopped when trigger exit().

Input parameters:

N/A

Return value:

N/A

6.2 Input and Output

The maximum length of input and output for the executable file in ROCKEY5 TIME is both 256 bytes.

6.2.1 pbInBuff and wInLen

#define pbInBuff ((BYTE xdata *)INPUT_DATA_OFFSET + 2)

#define wInLen (*(WORD xdata *)INPUT_DATA_OFFSET)

Objective:

Input data. C51 program receives data from external interface (PC side program) to process in ROCKEY5 TIME.

pbInBuff is a address pointing to the input data.

wInLen is the length of input data.

Input parameters:

N/A

Return value:

N/A

Example:

(1) In C51 program, to assign a value to a variable in the type of int, long, word, dword, float and double, the high-order byte is ordered first and then the low-order byte. By contrast, in PC, the low-order byte is order first other than the high-order byte, reversing the order is needed when inputing. To implement the reversing, an external function can be called or using C51 language “swap” interface introducted in 6.2.3.

The following code demonstrates inputing a digit (8) to C51 program, external language is C.

```
    ///////Reversing////////
    void FlipBuffer(unsigned char* pBuf, unsigned long ulLen)
    {
        unsigned char ucTemp;
        for(unsigned long i = 0; i < ulLen >> 1; ++i)
        {
            ucTemp = pBuf[i];
            pBuf[i] = pBuf[ulLen - i - 1];
            pBuf[ulLen - i - 1] = ucTemp;
        }
    }
```

```
        int main(int argc, char* argv[])
        {
            short sCData = 8;    //Memory: 0x08 0x00
            long lCData = 8;     //Memory: 0x08 0x00 0x00 0x00
            float fCData = 8.0;  //Memory: 0x00 0x00 0x00 0x41

        double dCData = 8.0;     //Memory: 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x40
        BYTE pbInData[18]={0};   //import short, long, float and double, 18 bytes in
total

        FlipBuffer((unsigned char*)&sCData, 2);//memory:0x00 0x08
        FlipBuffer((unsigned char*)&lCData, 4);//Memory: 0x00 0x00 0x00 0x08
        FlipBuffer((unsigned char*)&fCData, 4);//Memory: 0x41 0x00 0x00 0x00

        FlipBuffer((unsigned char*)&dCData, 8);
        //Memory: 0x40 0x20 0x00 0x00 0x00 0x00 0x00 0x00

        memcpy(pbInData, &sCData, 2);
        memcpy(pbInData+2, &lCData, 4);
        memcpy(pbInData+6, &fCData, 4);
        memcpy(pbInData+10, &dCData, 8);
        //import pbInData to ROCKEY5 TIME, 18 bytes.
        }
```

In C51:

```
        if(wInLen != 18)
               _exit();          //if the length is over 18 bytes

        //////C51 int type is 2 bytes, but in C is short.
        int xdata iData;
    //assign digit 8 to iData, the serial number for first two byte in pbInBuff is 0,1
        memcpy(&iData, pbInBuff, 2);

        /////C51 long type is identical to C, 4 bytes
        long xdata lData;
        // assign digit 8 to iData, the serial number in pbInBuff is from 2 to 5, 4 bytes.
        memcpy(&lData, pbInBuff + 2, 4);

        ////// C51 float type is identical to C, 4 bytes/////
        float xdata fData;
        // assign digit 8.0 to iData the serial number in pbInBuff is from 2 to 5, 4 bytes
        memcpy (&fData, pbInBuff + 6, 4);

        ///// C51 Double type is 8 bytes
        DOUBLE xdata dData;
        //assign digit 8.0 to iData the serial number in pbInBuff is from 10 to 17, 8 bytes
        memcpy (&dData, pbInBuff + 10, 8);
```

(2) The reversing is not required when assign to string type

In C:

```
        int main(int argc, char* argv[])
        {
            //including the string ending 0x48 0x45 0x4C 0x4C 0x4F 0x00
        char cBuffer[6] = “HELLO”;
        }
```

In C51:

```
        char xdata buffer[250];
        memcpy(buffer, pbInBuff, 6);
        //assign 6 bytes (“HELLO”, 0x48 0x45 0x4C 0x4C 0x4F 0x00) to buffer
```

(3) Assign value to multiple variables, for instance: assign long, char array and double to C51 program.

In C:

```
        int main(int argc, char* argv[])
        {
            BYTE pbInData[18] = {0};

            long lCData = 8;
            FlipBuffer((unsigned char*)&lCData, 4);

            char cBuffer[6] = “HELLO”;

            double dCData = 8.0;
            FlipBuffer((unsigned char*)&dCData, 8);

        memcpy(pbInData, &lCData, 4);
        memcpy(pbInData+4, cBuffer, 6);
        memcpy(pbInData+10, &dCData, 8);

        //import pbInData to ROCKEY5 TIME, 18 bytes
        }
```

In C51:

```
        long xdata lData;
        char xdata buffer[250];
        double xdata dData;

        if(wInLen != 18)
                   _exit();          //if the length is over 18 bytes

        memcpy(&lData, pbInBuff, 4);
        // assign digit 8 (0x00 0x00 0x00 0x08) to iData, the serial number in pbInBuff
is from //0 to 3, 4 bytes

        memcpy(buffer, pbInBuff + 4, 6);
        //assign “HELLO” (0x48 0x45 0x4C 0x4C 0x4F 0x00) to buffer
        // the serial number in pbInBuff is from 4 to 9, 6 bytes
        memcpy(&dData, pbInBuff + 10, 8);
        //assign 8.0 (0x40 0x20 0x00 0x00 0x00 0x00 0x00 0x00) to dData
        // the serial number in pbInBuff is from 10 to 17, 8 bytes
```

6.2.2 set_response

BYTE _set_response(WORD wLen, void *pvData)

Objective:

Get output data. After the program has run out of ROCKEY5 TIME, the function returns the result to the external program. Note that the maximum length is 256 bytes.

Input parameters:

wLeb	[in]the length of output data
*pvdata	[in] the address of output data

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example

(1) When output data is in type of int, long, word, dword, float and double, the reversing process is needed due to the sequence of C51 program storage. To implement the reversing, an external function can be called or using C51 language swap interface introduced in section 6.2.3.

In C51:

```
        int xdata a;
        a = 1+2;
        _set_response(2,&a); //Memory: 0x00 0x03
```

In C:

```
    ////////reversing function//////
    void FlipBuffer(unsigned char* pBuf, unsigned long ulLen)
    {
        unsigned char ucTemp;
        for(unsigned long i = 0; i < ulLen >> 1; ++i)
        {
            ucTemp = pBuf[i];
            pBuf[i] = pBuf[ulLen - i - 1];
            pBuf[ulLen - i - 1] = ucTemp;
        }
    }

    int main(int argc, char* argv[])
    {
        int i = 0;
        ……
        //assign the result from ROCKEY4 TIME to i, the memory of i now is 0x00 0x03

    FlipBuffer((unsigned char*)&i, 2); // after reversing: 0x03 0x00
    printf(“%d\n”,i); // The print result is as expected 3
    }
```

(2) The string does not need reversing.

In C51:

```
    char xdata buffer[] = “HELLO”;
    _set_response(6, buffer); //Pass “HELLO”
```

In C:

```
    int main(int argc, char* argv[])
    {
        char cBuffer[1024];

        //pass the result from ROCKEY5 TIME to cBuffer
    printf(“%s\n”, cBuffer); // The output print is “HELLO”
    }
```

6.2.3 swap

void _swap(void* pvData,unsigned short wLen)

Objective:

Reverse the data sequence. When import the data in type of int, long, word, dword, float and double, reversing is required. Sugguestion: To enhance the performance, reversing process is better running in external program.

Input parameters:

*pvData	[in] the storage address for reversing data
wLen	[in] the length of data

Return value:

N/A

Example:

```
    char xdata buffer[] = “HELLO”;
    _swap(buffer, 5); //After reversing buffer is : "OLLEH”
```

6.3 File Operation

6.3.1 create

BYTE _create(

WORD wFileID,

WORD wSize,

BYTE bFileType,

BYTE bFlag,

HANDLE *pHandle)

Objective:

Create file. The user is enabled to create the data file, executable file and key file in ROCKEY5 TIME.

Input parameters:

wFileID	[in] The file ID, 2 bytes.
wSize	[in] The size of the file, 2 bytes.
bFileType	[in] The file type, 1 byte, for more detail check Table 6-1
bFlag	[in] The file flag byte, 1 byte, check Table 6-2
*pHandle	[out] File handle. If the file is open, it returns the file handle.

Name	Value	Explationation
FILE_TYPE_EXE	0x00	Executable file (Wrt)
FILE_TYPE_DATA	0x01	Internal data file (R/W)
FILE_TYPE_RSA_PUB	0x02	RSA public key file(R/W)
FILE_TYPE_RSA_SEC	0x03	RSA private key (Wrt)

Table 6-1 File Type

Check Table 6-1 for more about file type. In C51 program (the executable file in ROCKEY5 TIME). And the file properties are in following:

Excutable file: enable to be modified by the other executable file, but impossible to be read in any circumstances.

Internal data file: enable to be read or written by executable file.

RSA public key file: enable to be read or written by executable file.

RSA private key file: enable to be modified by executable file, but impossible to be read in any circumstances.

CREATE_OPEN_ALWAYS	0x00	If the file already exists, it opens file otherwise creates and then opens
CREATE_FILE_NEW	0x01	Create and open the new file, if the

		file already exists, it returns an error.
CREATE_OPEN_EXISTING	0x02	Open the existed file, similar to _open()

Table 6-2 File Flags

Return value:

Success returns 0. Other return means failure, for more detail check the error list in section 6.12.

Example:

```
    HANDLE xdata hFile = 0;
    BYTE xdata bRes = 0;

    //Create a data file of 16 bytes （0x10）and ID as 0x1008
    bRes=_create(0x1008,0x10,FILE_TYPE_DATA,CREATE_OPEN_ALWAYS,&hFile);

    if(bRes != 0)
    {
        _set_response(1,&bRes);
        _exit();
    }
```

6.3.2 open

BYTE _open(WORD wFileID, HANDLE *pHandle)

Objective:

Open file. Open the file in ROCKEY5 TIME.

- The data file and public key file can be read or written when opened.
- The private key file and executable file can be only written rather than read.

Input parameters:

wFileID	[in] File ID, 2 bytes
*pHandle	[out] Return file handle

Return value:

Success returns 0. Other return means failure, for more detail check the error list in section 6.12.

Example

```
    word xdata wFid = 0x1008;
    HANDLE xdata hFile = 0;
    BYTE xdata bRes = 0;

    bRes  = _open(wFid, &hFile); // Open file Id is 0x1008
    if(bRes != 0)
    {
        _set_response(1,&bRes);
        _exit();
    }
```

6.3.3 close

BYTE _close(HANDLE handle)

Objective:

Close file. To complete the operation of file, close function is needed. Note that a prerequisite to close the file is to open the file.

Input parameters:

handle [in]File handle which is retrieved by calling open function.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
    word xdata wFid = 0x1008;
    HANDLE xdata hFile = 0;

    _open(wFid, &hFile); // Open a file ID is 0x1008
        ……
    _close(hFile);       // Close a file, ID is 0x1008
```

6.3.4 read

BYTE _read(HANDLE handle, WORD wOffset, BYTE bLen, void *pvData)

Objective:

Read file. This function can only be called for data file and public key file. For the executable file, it cannot be read. Note that a prerequisite to read the file is to open the file.

Input parameters:

handle [in] File handle which is retrieved by calling open function

wOffset [in] Offset value. The positon to read the file from

bLen [in] The length of data to be read

*pvData [out] The output data address.

Return value:

Success returns 0. Other return means failure. For more detail, check the list in section 6.12.

Example:

```
    word xdata wFid = 0x1008;
    HANDLE xdata hFile = 0;
    char xdata buffer[250];
    BYTE xdata bRes = 0;

    bRes = _open(wFid, &hFile); // Open a file whose ID is 0x1008
    if(bRes != 0)
    {
        _set_response(1,&bRes);
        _exit();
    }

    bRes = _read(hFile, 0, 10, buffer);
    //From ID ID 0x1008, read 10 bytes to buffer starting from 0

    if(bRes != 0)
    {
        _set_response(1,&bRes);
        _exit();
    }

    _close(hFile);      //close the file whose ID is 0x1008
```

6.3.5 write

BYTE _write(HANDLE handle, WORD wOffset, BYTE bLen,const void *pvData)

Objective:

Write file. Write data into the file. The function is mainly used for data file, public key file and executable file. Note that a prerequisite to call this function is to open the file.

Input parameters:

handle	[in] File handle. Use function open to get the handle
wOffset	[in] Offset value. The position to write the file from
bLen	[in] The length of data to be written
*pvData	[in] Store the address of data to be written

Return value:

Success returns 0. Other return means failure. For more detail, check the list in section 6.12.

Example:

```
    word  xdata wFid = 0x1008;
    HANDLE xdata hFile = 0;
    char xdata buffer[] = “HELLO”;
    BYTE xdata bRes = 0;

    bRes = _open(wFid, &hFile); // Open a file whose ID is 0x1008
    if(bRes != 0)
    {
        _set_response(1,&bRes);
        _exit();
    }

    bRes = _write(hFile, 0, sizeof(buffer), buffer);
    //write “Hello” to ID 0x10088, starting from 0 byte of this file.

    if(bRes != 0)
    {
        _set_response(1,&bRes);
        _exit();
    }
    _close(hFile);       // Close the file whose ID is 0x10088
```

6.3.6 get_file_infor

BYTE _get_file_infor(PEFINFO pFileInfo)

Objective:

Get the file property information

Input parameters:

pFileInfo [out]The point of the File information structure which includes File ID, File Type and size.

EFINFO file information structure defines as following:

```
typedef struct _FILE_INFO

{

    WORD    wFileID;    // ID

    BYTE  bFileType;    // file type, please check table 6-1 for details

    WORD    wFileSize;  // file size

}

EFINFO,*PEFINFO;
```

Return value:

Success returns 0. Other return means failure. For more detail, check the list in section 6.12.

Example:

```
    EFINFO xdata fInfo;
    BYTE xdata bRes = 0;

    fInfo.wFileID = 0x1008;
    bRes = _get_file_infor(&fInfo); //Get ID 0x1008 information
```

```
        if(bRes != 0)
        {
                _set_response(1,&bRes);
                _exit();
        }
```

6.4 Algorithms

A brief introduction is given for the cryptography conception.

- **Symmetric encryption algorithms**

Symmetric encryption algorithms (**Symmetric-key encryption algorithms**) use the same key for encryption and decryption. Symmetric-key encryption algorithms can be divided into Stream algorithms (Stream ciphers) and Block algorithms (Block ciphers).

- **ECB and CBC**

The simplest of the encryption modes is the **electronic codebook** (ECB) mode. The message is divided into blocks and each block is encrypted separately.

In the **cipher-block chaining** (CBC) mode, each block of plaintext is XORed with the previous cipher text block before being encrypted. This way, each cipher text block is dependent on all plaintext blocks processed up to that point. Also, to make each message unique, an initialization vector must be used in the first block.

- **Padding**

Many classical ciphers arrange the plaintext into particular patterns (e.g., squares, rectangles, etc) and if the plaintext doesn't exactly fit, it is often necessary to supply additional letters to fill out the pattern.

- **Cryptographic hash function**

A **cryptographic hash function** is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, the (**cryptographic**) **hash value**, such that an accidental or intentional change to the data will change the hash value. The data to be encoded is often called the "message," and the hash value is sometimes called the **message digest** or simply **digests.**

- **Asymmetric cryptography**

Asymmetric cryptography refers to a cryptographic system requiring two separate keys, one to lock or encrypt the plaintext, and the other to unlock or decrypt the cipher text. Neither key will do both functions. One of these keys is published or public and the other is kept private. If the lock/encryption key is the one published then the system enables private communication from the public to the unlocking key's owner. If the unlock/decryption key is the one published then the system serves as a signature verifier of documents locked by the owner of the private key.

ROCKEY5 TIME supports hardware implemented RSA 512/1024/2048 bits RSA calculation.

- **Digital signature**

A **digital signature** is a mathematical scheme for demonstrating the authenticity of a digital message or document. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, and that it was not altered in transit. Digital signatures are commonly used for software distribution, financial transactions, and in other cases where it is important to detect forgery or tampering.

6.4.1 des_enc

BYTE _des_enc(const void *pvKey, BYTE bLen, void *pvData)

Objective:

Encrypt data by DES. It requires messages whose length is a multiple of the block size (8 bytes), or messages have to be padded to bring them to this length. The encrypt mode is ECB and for the other mode like CBC needs to be developed by oneself.

Input Parameters:

*pvKey	[in]The address of DES key whose length is 8 bytes.
bLen	[in]The length of encrypted data(a multiple of 8 bytes).
*pvData	[in/out]The plain text as input and cipher text as output.

Return value:

Success returns 0. Other return means failure. For more detail, check the list in section 6.12.

Example:

```
BYTE xdata bRes = 0;
// the plain text needs to be encrypted as a multiple of 8
char xdata text[8] = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07};
// DES key, 8 bytes
BYTE xdata deskey[8] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88};
bRes = _des_enc(deskey, 8, text);
if(bRes != 0)
{
    _set_response(1,&bRes);
    _exit();
}

//encrypted text：0x9A 0xB9 0xB1 0x88 0xB1 0x6A 0x62 0x40
_set_response(8,text);
_exit();
```

6.4.2 des_dec

BYTE _des_dec(const void *pvKey, BYTE bLen, void *pvData)

Objective:

Decrypt data by DES. It requires messages whose length is a multiple of the block size (typically 8 or 16 bytes), so messages have to be padded to bring them to this length. The encrypt mode is ECB and for the other mode like CBC needs to be developed by oneself.

Input parameters:

*pvKey	[in]The address of DES key whose length is 8 bytes.
bLen	[in]The length of decrypted data (a multiple of 8 bytes).
*pvData	[in/out]The cipher text as input and plain text as output.

Return value:

Success returns 0. Other return means failure. For more detail, check the list in section 6.12.

Example:

```
    BYTE xdata bRes = 0;
    // the plain text needs to be decrypted as a multiple of 8
    char xdata text[8] = {0x9A, 0xB9, 0xB1, 0x88, 0xB1, 0x6A, 0x62, 0x40,};
    // DES key, 8 bytes
    BYTE xdata deskey[8] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88};
    bRes = _des_dec(deskey, 8, text);
    if(bRes != 0)
    {
            _set_response(1,&bRes); // duplicate buffer
            _exit();
    }

    // decrypted text：0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
    _set_response(8,text);
    _exit();
```

6.4.3 aes_enc

extern BYTE _aes_enc(const void *pvKey, BYTE bLen, void *pvData);

Objective:

Encrypt data by AES algorithm.

Note:

AES key is 16 bytes and it requires messages whose length is a multiple of the block size (16 bytes), or messages have to be padded to bring them to this length. The encrypt mode is ECB and for the other mode like CBC needs to be developed by oneself.

Input parameters:

*pvKey	[in] The address of AES key whose length is 16 bytes
bLen	[in] The length of encrypted data (a multiple of 16 bytes).

*pvData [in/out] The plain text as input and cipher text as output.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
       BYTE xdata bRes = 0;
       // the plain text needs to be encrypted as a multiple of 16
       Char xdata text[16] =
       {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0
x0f};
       //AES key, 16 bytes
       BYTE xdata aeskey[16] =
       {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0xaa,0xbb,0xcc,0xdd,0xee,0
xff};
       bRes = _aes_enc(aeskey, 16, text); //duplicate buffer
       if(bRes != 0)
       {
           _set_response(1,&bRes);
           _exit();
       }
       _set_response(16,text);
       _exit();
```

6.4.4 aes_dec

extern BYTE _aes_dec(const void *pvKey, BYTE bLen, void *pvData);

Objective:

Decrypt data by AES

Input parameters:

*pvKey [in/out] The 16 bytes buffer for storing AES key.

bLen [in] The cipher data length

*pvData [in/out] Pointer points to the cipher text buffer.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
       BYTE xdata bRes = 0;
       //the cipher text needs to be decrypted as a multiple of 16
       Char xdata text[16] =
       {0x9A,0xB9,0xB1,0x88,0xB1,0x6A,0x62,0x40,0x8A,0x52,0x49,0x96,0x8A,0XB8,0XB2,0
x66};
       // AES key, 16 bytes
       BYTE xdata aeskey[16] =
       {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0xaa,0xbb,0xcc,0xdd,0xee,0
xff};
       bRes = _aes_dec(aeskey, 16, text); // duplicate buffer
       if(bRes != 0)
       {
           _set_response(1,&bRes);
           _exit();
       }

       _set_response(16,text);
       _exit();
```

6.4.5 tdes_enc

BYTE _tdes_enc(const void *pvKey, BYTE bLen, void *pvData)

Objective:

3DES encryption function. 3DES key is 16 bytes and it requires messages whose length is a multiple of the block size (16 bytes). Or messages have to be padded to bring them to this length. The encrypt mode is ECB and for the other mode like CBC needs to be developed by oneself.

Input parameters:

*pvKey	[in] The 16 bytes buffer store 3DES key.
bLen	[in] The cipher data length
*pvData	[in/out] The plain text as input and cipher text as output.

Return value:

Success returns 0. Other return means failure. For more detail, check the list in section 6.12.

Example:

```
        BYTE xdata bRes = 0;
        // the plain text needs to be encrypted as a multiple of 8
        char xdata text[8] = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07};
        // 3DES, 16 bytes
        BYTE xdata tdeskey[16] = {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
                                   0x88,0x99,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF};
        bRes = _tdes_enc(tdeskey, 8, text); // duplicate buffer
        if(bRes != 0)
        {
            _set_response(1,&bRes);
            _exit();
        }

        // encrypted text：0x5D 0x99 0x07 0x87 0xB0 0x67 0x37 0x87
        _set_response(8,text);
        _exit();
```

6.4.6 tdes_dec

BYTE _tdes_dec(const void *pvKey, BYTE bLen, void *pvData)

Objective:

3DES decryption function. 3DES key is 16 bytes and it requires messages whose length is a multiple of the block size (16 bytes), so messages have to be padded to bring them to this length. The encrypt mode is ECB and for the other mode like CBC needs to be developed by oneself.

Input parameters:

*pvKey	[in] The 16 bytes buffer store 3DES key
bLen	[in] The plain data length
*pvData	[in/out] The cipher text as input and plain text as output.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
BYTE xdata bRes = 0;
// the cipher text needs to be decrypted as a multiple of 8
char text[8] = {0x5D, 0x99, 0x07, 0x87, 0xB0, 0x67, 0x37, 0x87};
//3DES key, 16 bytes
BYTE xdata tdeskey[16] = {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
                           0x88,0x99,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF};
bRes = _tdes_dec(tdeskey, 8, text); // duplicate buffer
if(bRes != 0)
{
    _set_response(1,&bRes);
    _exit();
}

// the decrypted text：0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
_set_response(8,text);
_exit();
```

6.4.7 sha1_init

BYTE _sha1_init(PSHA_CONTEXT pCtx)

Objective:

The initialaztion is required before using SHA1

Input parameters:

pCtx	[in] The address of the environment context

SHA_CONTEXT: Environment context structure

```
typedef struct _tagSHA_CONTEXT
{
        DWORD     h[5];
        DWORD   dwTotalLength;
        BYTEbRemainLength;
        BYTEpbRemainBuf[ROCKEY5_SHA_CBLOCK];
}SHA_CONTEXT,*PSHA_CONTEXT;
```

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

Check the example of sha1_final.

6.4.8 sha1_update

BYTE _sha1_update(PSHA_CONTEXT pCtx,const void *pvData, BYTE bLen)

Objective:

SHA1 hash algorithm

Input Parameters:

pCtx	[in] The address of the environment context
*pvData	[in] The hash data
bLen	[in] The length of hash data

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

Check the example of sha1_final.

6.4.9 sha1_final

BYTE _sha1_final(PSHA_CONTEXT pCtx, void *pvResult)

Objective:

Result of SHA-1. 20 bytes long.

Input parameters:

pCtx [in] The address of the environment context

*pvResult [out] The address of hash result

Note:

The hash result is saved in COS (check section 3.2.9), a buffer in COS saves the result from last hashing.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
BYTE  xdata bRes = 0;
char xdata result[20];
BYTE xdata text[] = "HELLO";
SHA_CONTEXT xdata sha1ctx;

// The address of the environment context
bRes = _sha1_init(&sha1ctx);
if(bRes != 0)
{
    _set_response(1,&bRes);
    _exit();
}

// Hashing
bRes = _sha1_update(&sha1ctx, text, 5);
if(bRes != 0)
{
    _set_response(1,&bRes);
    _exit();
}

// Get the result with length of 20 bytes
bRes = _sha1_final(&sha1ctx, result);
if(bRes != 0)
{
    _set_response(1,&bRes);
    _exit();
}

_set_response(20,result);
_exit();
```

6.4.10 md5_init

BYTE _md5_init(PMD5_CONTEXT pCtx)

Objective:

The initialization is required before using MD5

Input parameters:

pCtx [in] The address of the environment context

MD5_CONTEXT: Environment context structure

```
typedef struct _tagMD5_CONTEXT
{
        DWORD h[4];
        DWORD dwTotalLength;
        BYTE    bRemainLength;
        BYTE    pbRemainBuf[ROCKEY5_MD5_CBLOCK];
}MD5_CONTEXT,*PMD5_CONTEXT;
```

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

Check md5_final example.

6.4.11 md5_update

BYTE _md5_update(PMD5_CONTEXT pCtx,const void *pvData, BYTE bLen)

Objective:

Perform MD5 calculation.

Input parameters:

pCtx	[in] The address of the environment context
*pvData	[in] Hash data
bLen	[in] Hash data length

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

Check md5_final example.

6.4.12 md5_final

BYTE _md5_final(PMD5_CONTEXT pCtx, void *pvResult)

Objective:

Get MD5 result. The result is 16 bytes long

Input parameters:

pCtx [in] The address of the environment context

*pvResult [in]The address of hash result

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
        BYTE  xdata bRes = 0;
        char xdata result[16];
        BYTE xdata text[] = "HELLO";
        MD5_CONTEXT xdata md5ctx;

        //intial the environment context before hashing
        bRes = _md5_init(&md5ctx);
        if(bRes != 0)
        {
            _set_response(1,&bRes);
            _exit();
        }

        //hasing
        bRes = _md5_update(&md5ctx, text, 5);
        if(bRes != 0)
        {
            _set_response(1,&bRes);
            _exit();
        }

        //Get MD5 hashing result with 16 bytes
        bRes = _md5_final(&md5ctx, result);
        if(bRes != 0)
        {
            _set_response(1,&bRes);
            _exit();
        }

        _set_response(16,result);
        _exit();
```

6.4.13 rsa_enc

BYTE _rsa_enc(BYTE bMode, WORD wFileID, WORD wLen, void *pvData)

Objective:

Encrypt by using RSA public key.

Input parameters:

bMode	[in]RSA encryption. Check Table 6-3, By default is 1024 bit.
wFileID	[in]Public key ID
wLen	[in]The length of plain text
*pvData	[in/out]The plain text as input and cipher text as output.

Note:

- bMode：RSA encryption. There are two modes for RSA: RSA_CALC_NORMAL and RSA_CALC_PKCS. Both of them require to use with RSA key bit (RSA_CALC_NORMAL | RSA_CALC_BIT_1024) and default bit is 1024.

RSA encryption		
Name	**Value**	**Explaination**
RSA_CALC_NORMAL	0x00	Without need of coding.
RSA_CALC_PKCS	0x02	PKCS#1 stardard encryption
RSA key bit		
RSA_CALC_BIT_512	0x10	Encrypt with RSA 512 bit private key
RSA_CALC_BIT_1024 (default)	0x00	Encrypt with RSA 1024 bit private key
RSA_CALC_BIT_2048	0x20	Encrypt with RSA 2048 bit private key

Table 6-3

- The input parameter is in fixed length when using RSA_CALC_NORMAL as in the following Table 6-4. If the plain text is not in fixed length, it is padded or blocked to meet the length. The interface can pad 0 in front of the value if it is not padded.

RSA_CALC_NORMAL plain text length		
Pattern	**Value**	**Plain text length (bytes)**
RSA_CALC_BIT_512	0x10	64

RSA_CALC_BIT_1024 (default)	0x00	128
RSA_CALC_BIT_2048	0x20	256

Table 6-4 RSA_CALC_NORMAL plain text length

- The input parameter is in fixed length when using RSA CALC PKCS as in the following Table 6-5.

RSA_CALC_PKCS plain text length		
Pattern	**value**	**Plain text length (bytes)**
RSA_CALC_BIT_512	0x10	1-53
RSA_CALC_BIT_1024 (default)	0x00	1-117
RSA_CALC_BIT_2048	0x20	1-245

Table 6-5 RSA_CALC_PKCS plain text length

- The cipher text calculated by RSA is fixed length. Enough space needs to be allocated for *pvData buffer

Cipher text length		
Name	**Value**	**The Length of cipher (bytes)**
RSA_CALC_BIT_512	0x10	64
RSA_CALC_BIT_1024(default)	0x00	128
RSA_CALC_BIT_2048	0x20	256

Table 6-6 cipher text length

- To encrypt the data in RSA, the plain text is required to be shorter than the public key. In order to deal with this situation, the first byte of plain text is set to be 0. *pvData points to the byte whose value is 0.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12 .

Example:

Check rsa_gen_key example.

6.4.14 rsa_dec

BYTE _rsa_dec(BYTE bMode, WORD wFileID, WORD wLen, void *pvData)

Objective:

Decrypt by using RSA private key

Input parameters:

bMode	[in] RSA encryption mode, plese refer to Table 6-7. 1024 bit by default
wFileID	[in]Private key ID
wLen	[in]The length of the cipher text.
*pvData	[in/out] The cipher text as input and plain text as output.

Note:

- bMode: RSA encryption mode, check 6.4.11 rsa_enc for more detail.
- wLen: The input parameter is in fixed length.If the plain text is not in fixed length, it is padded or blocked to meet the length. The reverse operation is required when decrypt the cipher text.

cipher text length		
Name	**Value**	**The Length of cipher (byte)**
RSA_CALC_BIT_512	0x10	64
RSA_CALC_BIT_1024(default)	0x00	128
RSA_CALC_BIT_2048	0x20	256

Table 6-7 cipher text length

- The first byte of the plain text decrypted by RSA_CALC_PKCS is the plain text length and the rest are the plain text.

Retuen value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

Check the example of rsa_gen_key.

6.4.15 rsa_gen_key

BYTE _rsa_gen_key(WORD wPubID, WORD wPriID,WORD wKeyBitLen)

Objective:

Generate RSA key pair

Input parameters:

wPubID	[in]Public key ID
wPriID	[in]Private key ID
wKeyBitLen	[in]The length of RSA key (512/1024/2048)

Note:

■Considering the encrypt strength and operational speed, the common length of RSA private key is 1024 bit. The different private key length maps to the related data check 6.4.11 and 6.4.12 for more detail.

■wPubID and wPriID must use unoccupied ID in the current directory. This function will create two new file.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
        char xdata plaintext[] = "HELLO";
            BYTE xdata ciphertext[128];
            BYTE xdata bRes = 0;
            WORD xdata wPubKeyFileID = 0x1002;   //Public key ID
            WORD xdata wPriKeyFileID = 0x1004;   //Private key ID
            WORD xdata wKeyLen = 1024;           //RSA key length

            //Generate key
            bRes = _rsa_gen_key(wPubKeyFileID, wPriKeyFileID, wKeyLen);
            if(bRes != 0)
            {
                _set_response(1,&bRes);
                _exit();
            }

            memset(ciphertext,0,128);       //Assign 0 to cipher text
            //Copy HELLO”  to cipher text
            //Set the first byte as 0 in case that the plain text is over key length.
        memcpy(ciphertext+1, plaintext, 6);
            //RSA encryption, the 128 bytes result is in cipher text
            bRes = _rsa_enc(RSA_CALC_PKCS | RSA_CALC_BIT_1024, wPubKeyFileID, 6,
ciphertext);
            if(bRes != 0)
            {
                _set_response(1,&bRes);
                _exit();
            }

            //RSA decrypts with length of 128 bytes.
            bRes = _rsa_dec(RSA_CALC_PKCS | RSA_CALC_BIT_1024, wPriKeyFileID, 128, 
ciphertext);
            if(bRes != 0)
            {
                _set_response(1,&bRes);
                _exit();
            }

            _set_response(128, ciphertext);
            _exit();
```

6.4.16 rsa_sign

BYTE _rsa_sign(BYTE bMode, WORD wFileID, WORD wLen, void *pvData)

Objective:

Sign the data by using RSA private key

Input parameters:

bMode	[in]Signature mode. Check Table 6-8.
wFileID	[in]Private Key ID
wLen	[in]The length of the signing data
*pvData	[in/out]Input is the data to be signed and output is the signed data.

Note:

- The signature generating requires hash computing. The hash algorithm for this function is SHA1 and generates 20 bytes output data. The signature requires following PKCS#1 standard, padding 108 bytes （0x00，0x01，0xFF….，0x00）in front of the hashed value (20 bytes), and then going to be signed. In order to meet the Microsoft CAPI standard, padding 15 bytes （0x30, 0x21, 0x30, 0x09, 0x06, 0x05, 0x2b, 0x0e, 0x03, 0x02, 0x1a, 0x05, 0x00, 0x04, 0x14）instead. And then pad 93 bytes （0x00，0x01，0xFF…90 times，0x00）to sign. There are four signature modes. Each mode should be used with a matched RSA key length. Please refer to the table below.

RSA signature mode			
Name	**Value**	**Description**	**The length of data to be signed(bytes)**
RSA_CALC_NORMAL	0x00	Sign the hash value from SHA1 by private key	20

RSA_CALC_HASH	0x01	Hash the imported data by SHA1, and then sign the value	Data length needs to be shorter than 2k which is the card memory size.
RSA_CALC_PKCS	0x02	Sign the hash value from SHA1 following PKCS#1 standard.	20
RSA_CALC_HASH|RSA_CALC_PKCS	0x03	Hash the imported data by SHA1, and then sign the value following PKCS#1 standard.	Data length needs to be shorter than 2k which is the card memory size.
RSA private key bit			
RSA_CALC_BIT_512	0x10	Sign with 512 bit private key	
RSA_CALC_BIT_1024 (default)	0x00	Sign with 1024 bit private key	
RSA_CALC_BIT_2048	0x20	Sign with 2048 bit private key	

Table 6-8 Signature mode

■The export signed data differs in terms of the length of private key. The *pvData buffer needs to be allocated for space.

RSA signed data length	
RSA private key	**The signed data length**
512	64
1024	128
2048	256

Table 6-9 RSA signed data length

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

Check rsa_verify example.

6.4.17 rsa_verify

BYTE _rsa_verify(BYTE bMode, WORD wFileID, WORD wLen, void *pvData)

Objective:

Verify the signature by RSA public key.

Input parameters:

bMode	[in]Signature mode
wFileID	[in]Public key ID
wLen	[in]The length of the signature (64/128/256)
*pvData	[in]The signed data.

Note:

■ If the sign mode is RSA_CALC_NORMAL and RSA_CALC_HASH when using rsa_sign to sign, both of the two modes sign the hash value calculated by SHA1. The verification mode is RSA_CALC_NORMAL.

■ If the sign mode is RSA_CALC_PKCS or RSA_CALC_HASH|RSA_CALC_PKC, the verification mode is RSA_CALC_PKCS

■ The buffer in COS stores the hash value from the previous computation.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
        char xdata text[] = "HELLO";
        BYTE xdata signtext[128];
        BYTE xdata bRes = 0;
        WORD xdata wPubKeyFileID = 0x1002; //public key ID
        WORD xdata wPriKeyFileID = 0x1004; //private key ID

        memset(signtext,0,128);
        memcpy(signtext,text,6);

        //First has the value by SHA1 and sign
    //then sign the value following PKCS#1 standard and return 128 byte result.
        bRes=_rsa_sign(RSA_CALC_HASH|RSA_CALC_PKCS, wPriKeyFileID,6,signtext);
        if(bRes != 0)
        {
            _set_response(1,&bRes);
            _exit();
        }
```

```
            //verification by RSA public key
            bRes = _rsa_verify(RSA_CALC_PKCS, wPubKeyFileID, 128, signtext);
            if(bRes != 0)
            {
                _set_response(1,&bRes);
                _exit();
            }

            //return the signed result
            _set_response(128, signtext);
            _exit();
```

6.5 System Functions

6.5.1 rand

BYTE _rand(void *pvData, BYTE bLen)

Objective:

Get the random number

Input parameters:

*pvData	[out]Get the address of random number.
bLen	[in]The length of the random number.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
            BYTE xdata bRes = 0;
            BYTE xdata bRand[8];

            memset(bRand, 0, 8);
            bRes = _rand(bRand,8);
            if(bRes != 0)
            {
                _set_response(1,&bRes);
                _exit();
            }

            _set_response(8, bRand);
            _exit();
```

6.5.2 get_version

BYTE _get_version(BYTE bFlag,void *pvData,BYTE bLen)

Objective:

Get hardware information

Input parameters:

bFlag [in] Flag. For the description of bFlag, please refer to Table 6-10.

*pvData [out]The buffer address of storing information

bLen [in]The buffer size

The hardware flag information	
GLOBAL_SERIAL_NUMBER	serial number
GLOBAL_CLIENT_NUMBER	Vendor ID
GLOBAL_COS_VERSION	COS version

Table 6-10 The hardware flag information

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
BYTE xdata bRes = 0;
BYTE xdata bSN[8];

memset(bSN, 0, 8);
bRes = _get_version(GLOBAL_SERIAL_NUMBER,bSN,8);
if(bRes != 0)
{
    _set_response(1,&bRes);
    _exit();
}

_set_response(8, bSN);
_exit();
```

6.6 Clock Function

6.6.1 get_time_limit

extern BYTE _get_time_limit(DWORD *value, BYTE *policy);

Objective:

Get the COS time limitation

Input parameters:

*value [out]The address of expired time or remaining hours, determinted by the type of COS expired time

*policy [out]Flag of data type. If 0, COS stored expiry date. If 1, COS stored remaining hours.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7 Double-precision Floating-point Calculation

C51 lanuage itself does not support the double-precision floating-point, but the ROCKEY 5 TIME does. The symbol like ＋,－,×,÷ can not be directly used when computing in double-precision floating-point, except using the interface from ROCKEY5 TIME. The expression of the double-precision floating-point in C51 differs from common PC language. For example, the double-precision floating-point of 1.0 in PC shows address as 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x3F, but in C51 shows as 0x3F, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 instead. A reversing process is needed.

We offer a tool (Double2Byte.exe) that can converts 2 types between double-precision floating-point and 8 byte array. Note that, to ensure the double-precision floating-point, please do not use DOUBLE_INIT function, assign the double-precision floating-point variable in array way for example: DOUBLE A, to assign A to 1.0, memcpy (&A, "\x3F\xF0\x00\x00\x00\x00\x00\x00", 8);

6.7.1 add

BYTE _add(DOUBLE result, DOUBLE x, DOUBLE y)

Objective:

The addition function of double-precision floating-point.

Input parameters:

result	[out]The result of addition function
x	[in] The augend.
y	[in] The.addend

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12 .

Example:

```
    BYTE xdata bRes = 0;
       DOUBLE xdata dx, dy, dResult;
       //dx 为 1.0 ,( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x3F )
    DOUBLE_INIT(dx, 1.0);
    //dy 为 2.0 ,( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40 )
       DOUBLE_INIT(dy, 2.0);
```

```
           bRes = _add(dResult, dx, dy);
           if(bRes != 0)
           {
               _set_response(1,&bRes);
               _exit();
           }

           //return double-precision floating-point 3.0 ( 0x00 0x00 0x00 0x00 0x00 0x00
0x08 0x40 )
           _set_response(8, &dResult);
           _exit();
```

6.7.2 sub

BYTE _sub(DOUBLE result, DOUBLE x, DOUBLE y)

Objective:

Substraction function of the double-precision floating-point.

Input parameters:

result	[out]The result of substraction function
x	[in] The minuend
y	[in] The subtrahend

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.3 mul

BYTE _mul(DOUBLE result, DOUBLE x, DOUBLE y)

Objective:

The multiplication function for the double-precision floating-point

Input parameters:

result	[out]The result of the mutiplication
x	[in]The multiplicand
y	[in]The multiplier

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.4 div

BYTE _div(DOUBLE result, DOUBLE x, DOUBLE y)

Objective:

Division function for double-precision floating-point

Input parameters:

result	[out]The result of the division
x	[in]The dividend
y	[in]The divisor

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.5 atan2

BYTE _atan2(DOUBLE result, DOUBLE x, DOUBLE y)

Objective:

Arctangent function for double-precision floating-point number

Input parameters:

result	[out]The result of arctangent function.
x	[in]The dividend
y	[in]The divisor

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.6 mod

BYTE _mod(DOUBLE result, DOUBLE x, DOUBLE pMod)

Objective:

Modulo function for double-precision floating-point number

Input parameters:

result	[out]The result of the modular operation
x	[in]The dividend
pMod	[in]The divisor

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.7 pow

BYTE _pow(DOUBLE result, DOUBLE x, DOUBLE pExp)

Objective:

The power operations of double-precision floating-point

Input parameters:

result	[out]The result of the power operations
x	[in]The base number
pExp	[in]The exponent number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.8 modf

BYTE _modf(DOUBLE remain, DOUBLE x, DOUBLE intpart)

Objective:

Divide the double-precision floating-point number into integer part and the decimal fraction part.

Input parameters：

remain	[out]The divided decimal fraction part.
x	[in]The divided number
intpart	[out]The integer part.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.9 frexp

BYTE _frexp(DOUBLE result, DOUBLE x, WORD *pExp)

Objective:

Divide the the double-precision floating-point into the exponentiation of 2 times of a decimal fraction

Input parameters:

result	[out]The decimal fraction part

x	[in]The divided number
*pExp	[out] Exponent number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.10 ldexp

BYTE _ldexp(DOUBLE result, DOUBLE x, WORD exp)

Objective:

The exponentiation of 2 times of double-precision floating-point number

Input parameters:

result	[out]The result
x	[in]The double-precision floating-point number
exp	[in]The exponent of 2

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.11 sin

BYTE _sin(DOUBLE result, DOUBLE x)

Objective:

The sine function

Input parameters:

result	[out]The result
x	[in]The radian value

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.12 cos

BYTE _cos(DOUBLE result, DOUBLE x)

Objective:

Cosine function

Input parameters:

result	[out] The result
x	[in] The radian value

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.13 tan

BYTE _tan(DOUBLE result, DOUBLE x)

Objective:

The tangent function

Input parameters:

result	[out]The result
x	[in]The radian value

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.14 asin

BYTE _asin(DOUBLE result, DOUBLE x)

Objective:

The arcsine function

Input parameters:

result	[out]The result
x	[in]The sine value.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.15 acos

BYTE _acos(DOUBLE result, DOUBLE x)

Objective:

The arc cosine function

Input parameters:

result	[out]The result

x [in]The cosine value

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.16 atan

BYTE _atan(DOUBLE result, DOUBLE x)

Objective:

The arctangent function.

Input parameters:

result	[out] The result
x	[in] The tangent

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.17 sinh

BYTE _sinh(DOUBLE result, DOUBLE x)

Objective:

The hyperbolic sine function.

Input parameters:

result	[out]The result
x	[in]The radian value

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.18 cosh

BYTE _cosh(DOUBLE result, DOUBLE x)

Objective:

The hyperbolic cosine function.

Input parameters:

result	[out]The result
x	[in]The radian value

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.19 tanh

BYTE _tanh(DOUBLE result, DOUBLE x)

Objective:

The hyperbolic tangent function

Input parameters:

result	[out]The result
x	[in]The radian value

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.20 ceil

BYTE _ceil(DOUBLE result, DOUBLE x)

Objective:

The smallest integer not less than the double-precision floating-point number

Input parameters:

result	[out]The result
x	[in]The double-precision floating-point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.21 floor

BYTE _floor(DOUBLE result, DOUBLE x)

Objective:

The just integer number not more than the double-precision floating-point number

Input parameters:

result	[out]The result of the function
x	[in]The double-precision floating-point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.22 abs

BYTE _abs(DOUBLE result, DOUBLE x)

Objective:

The absolute value

Input parameters:

result	[out]The result of the funciton
x	[in]The double-precision floating-point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.23 exp

BYTE _exp(DOUBLE result, DOUBLE x)

Objective:

The power with base number e and double-precision floating-point number as index number

Input parameters:

result	[out]The result of the function
x	[in]The index number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.24 log

BYTE _log(DOUBLE result, DOUBLE x)

Objective:

The natural logarithm of the double-precision floating-point number.

Input parameters:

result	[out]The result of the function
x	[in]The double-precision floating-point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.25 log10

BYTE _log10(DOUBLE result, DOUBLE x)

Objective:

Common logarithm of of double-precision floating-point number

Input parameters:

result	[out]The result of the function
x	[in]The double-precision floating-point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.7.26 sqrt

BYTE _sqrt(DOUBLE result, DOUBLE x)

Objective:

The square root of the double-precision floating-point

Input parameters:

result	[out]The result
x	[in]The double-precision floating-point number

Return value:

Success returns 0. Other return means failure. For more detail, check the errol list in section 6.12.

6.7.27 cmp

char _cmp(DOUBLE x, DOUBLE y)

Objective:

The comparsion of the two double-precision floating-point numbers

Input parameters:

x	[in]Double-precision floating-point number
y	[in]Double-precision floating-point number

Return value:

Return 0 representing equality. Returns -1 representing x<y, returns 1 representing x>y.

Example:

```
char xdata cRes = 0;
    DOUBLE xdata dx,dy;

    //dx is 1.0 ,( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x3F )
DOUBLE_INIT(dx, 1.0);
//dy is 2.0 ,( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40 )
    DOUBLE_INIT(dy, 2.0);

    cRes = _cmp(dx, dy);
    _set_response(1, &cRes);//return 0xFF
    _exit();
```

6.8 Single Precision Floating Point Calculation

Single Precision Floating Point presents identically as high level language on PC with 4 byte but reversing sequence. For example, the address of single precision floating point 3.0 is 0x00, 0x00, 0x40, 0x40, but in C51, it represents as 0x40 0x40 0x00 0x00. For better performance, we suggest that the reverse process operates in high level language.

6.8.1 addf

float _addf(float x, float y)

Objective:

The addication function of Single Precision Floating Point

Input parameters:

x	[in]The augend
y	[in]The addend

Return value:

Return the addition result.

Example:

```
float xdata fx;
    float xdata fy;
    float xdata fResult;

    fx = 1.0;
    fy = 2.0;
    fResult = _addf(fx, fy);

    //return 0x40 0x40 0x00 0x00,
    _set_response(4, &fResult);
    _exit();
```

6.8.2 subf

float _subf(float x, float y)

Objective:

The substraction function of Single Precision Floating Point

Input parameters:

x	[in]The minuend
y	[in]The subtrahend

Return value:

Return the result of the substraction

6.8.3 mulf

float _mulf(float x, float y)

Objective:

The multiplication of Single Precision Floating Point

Input parameters:

x	[in]The multiplicand
y	[in]The multiplier

Return value:

Return the result of multiplication

6.8.4 divf

float _divf(float x, float y)

Objective:

The division of Single Precision Floating Point

Input parameters:

x	[in]The dividend
y	[in]The divisor

Return value:

Return the result of division

6.8.5 atan2f

float _atan2f(float x, float y)

Objective:

Arctangent function of Single Precision Floating Point

Input parameters:

x	[in]The dividend
y	[in]The divisor

Return value:

Return the result of arctangent calculation

6.8.6 fmodf

float _fmodf(float x, float y)

Objective：

The modulo function of Single Precision Floating Point

Input parameters:

x	[in]The dividend
y	[in]The divisor

Return value:

The result of the modulo function

6.8.7 powf

float _powf(float x, float y)

Objective:

The power function of Single Precision Floating Point

Input parameters:

x	[in]The base number
y	[in]The index number

Return value:

The result of the power function

6.8.8 cmpf

char _cmpf(float x, float y)

Objective:

The comparsion of two Single Precision Floating Point

Input parameters:

x	[in]Single Precision Floating Point number
y	[in]Single Precision Floating Point number

Return value:

Return 0 representing equality. Returns -1 representing x<y, Returns 1 representing x>y.

Example:

```
    float xdata fx;
        float xdata fy;
        char  xdata cRes;

        fx = 1.0;
        fy = 2.0;
        cRes = _cmpf(fx, fy);

        _set_response(1, &cRes); //return 0xFF = -1
        _exit();
```

6.8.9 sinf

float _sinf(float x)

Objective:

The sine function

Input parameters:

x	[in]Radian value

Return value:

Return the result of the function

6.8.10 cosf

float _cosf(float x)

Objective:

The cosine function

Input parameters:

x	[in]Radian value

Return value:

Return the result of the function

6.8.11 tanf

float _tanf(float x)

Objective:

Tangent function

Input parameters:

x	[in]Radian value

Return value:

Return the result of the function

6.8.12 asinf

float _asinf(float x)

Objective:

Arctsine function

Input parameters:

x	[in]The sine value

Return value:

Return the result of the function

6.8.13 acosf

float _acosf(float x)

Objective:

Arccosine function

Input parameters:

x	[in]The cosine value

Return value:

Return the result of the function

6.8.14 atanf

float _atanf(float x)

Objective:

Arctangent function

Input parameters:

x [in]A tangent value

Return value:

Return the result of the function

6.8.15 sinhf

float _sinhf(float x)

Objective:

The hyperbolic sine function

Input parameters:

x [in]Radian value

Return value:

Return the result of the function

6.8.16 coshf

float _coshf(float x)

Objective:

The hyperbolic cosine function

Input parameters:

x [in]Radian value

Return value:

Return the result of the function

6.8.17 tanhf

float _tanhf(float x)

Objective:

The hyperbolic tangent function

Input parameters:

x	[in]Radian value

Return value:

Return the result of the function

6.8.18 ceilf

float _ceilf(float x)

Objective:

The just integer not less than the Single Precision Floating Point number

Input parameters:

x	[in]Single Precision Floating Point number

Return value:

Return the just integer not less than the Single Precision Floating Point number.

6.8.19 floorf

float _floorf(float x)

Objective:

The just integer not more than Single Precision Floating Point number

Input parameters:

x	[in]Single Precision Floating Point number

Return value:

Return the just integer not more than Single Precision Floating Point number

6.8.20 absf

float _absf(float x)

Objective:

The absolute value of Single Precision Floating Point number

Input parameters:

x	[in]Single Precision Floating Point number

Return value:

Return the absolute value of Single Precision Floating Point

6.8.21 expf

float _expf(float x)

Objective:

Calculate the power of a number with e as base number and Single Precision Floating Point as index

Input parameters:

x	[in]The index number

Return value:

Return the result of the function

6.8.22 logf

float _logf(float x)

Objective:

Natural logarithm of Single Precision Floating Point number

Input parameters:

x	[in]Single Precision Floating Point number

Return value:

Return the result of the function

6.8.23 log10f

float _log10f(float x)

Objective:

The common logarithm of Single Precision Floating Point number

Input parameters:

x	[in]Single Precision Floating Point number

Return value:

Return the result of the function

6.8.24 sqrtf

float _sqrtf(float x)

Objective:

The square root of Single Precision Floating Point number

Input parameters:

x [in]Single Precision Floating Point number

Return value:

Return the result of the function

6.9 Time Functions

ROCKEY5 TIME with built-in clock chip as independent timing system does not reply on the computer system time. This way enhances the encryption security greatly. The secure design for hardware ensures tampering internal time on chip is impossible. The program in dongle can not run unless the setup time is older than the time in dongle.

The time in ROCKEY5 TIME is UTC standard time and the time error is less than 10 minutes per year. For one who needs to change to the local time, one should set up by oneself. For example: get the timer chip clock as 10:00 AM, change it to Beijing local time (UTC +08:00), as 18:00.

The battery in dongle can work more than three years. Besides that, when the dongle connects to the computer, there is no electric consuming for the battery. Even through the battery is out of work, the function not related to timing can still work.

6.9.1 time

BYTE _time(time_t *ptime)

Objective:

Get the hardware clock time of dongle. The time is UTC standard time, to convert to other local time, the adjustment is needed. “time_t” is unsigned long type as well as its return time.

Parameters:

*ptime [out] The address of the read time

Note:

Return 4 bytes value in unsigned long type. Note that a reversing process is required due to the C51 return value sequence in opposite to PC return value.

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
    time_t xdata time;
    BYTE xdata ret;
```

```
        time = 0;
        ret = _time(&time); // Get the current chip clock time
        if(ret != 0)
        {
        _set_response(1, &ret);
        _exit();
        }

        /*
        //return time in tim_t type (unsigned long)
        //Note that a reversing process is required due to the C51 return value sequence
in opposite //to PC return value.
        _swap(&time,4);
        */

        _set_response(4,&time);
        _exit();
```

6.9.2 mktime

BYTE _mktime(time_t *ptime, RTC_TIME_T *ptm)

Objective:

Convert the struct RTC_TIME_T time to time_t type. Both of the values are UTC standard time.

Parameters:

*ptime	[out]The address of converted time_t time
*ptm	[in]The pointer of the struct RTC_TIME_T to be converted

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
        BYTE xdata ret;
        time_t xdata exptime;
        RTC_TIME_T xdata exp = {0,0,0,20,0,5,110};/* 2010.6.20 */

        //convert the expired time to time_t type
        ret = _mktime(&exptime,&exp);
        if(ret != 0)
        {
        _set_response(1, &ret);
        _exit();
        }

        //Check if the dongle time is order than the setup time, if it is expired, returns
error
        if(time>exptime)
        {
        //if time is expired return error
        _set_response(1, &ret);
        _exit();
        }
        //if it is OK, run the rest code
        _set_response(4,&time);
        _exit();
```

6.9.3 gmtime

BYTE _gmtime(time_t *ptime, RTC_TIME_T *ptm)

Objective:

Convert the time_t time into RTC_TIME_T time. Both of them are UTC standard time.

Parameters:

*ptime	[in]The address of time_t time to be converted
*ptm	[in]The converted RTC_TIME_T time

Return value:

Success returns 0. Other return means failure. For more detail, check the errol list in section 6.12.

Example:

```
        BYTE xdata ret;
        time_t xdata time;
        RTC_TIME_T xdata stime;
        char xdata a[64];
        char xdata b[64];
        int xdata second;
        int xdata minute;
        int xdata hour;
        int xdata day;
        int xdata week;
        int xdata month;
        int xdata year;

        //convert time_t time to RTC_TIME_T time and return
        ret = _gmtime(&time,&stime);
        if(ret != 0)
        {
        _set_response(1, &ret);
        _exit();
        }
        second = stime.second;
        minute = stime.minute;
        hour = stime.hour;
        day = stime.day;
        week = stime.week;
        month = stime.month;
        year = stime.year;
        sprintf(a, "%.2d-%.2d-%.2d", year + 1900, month + 1 , day);
        sprintf(b, " %.2d:%.2d:%.2d", hour, minute, second);
        strcat(a, b);
        _set_response(20, a);
        _exit();
```

6.10 Type Conversion

This sector provides API for C51 proccessing data type conversion.

The aforemention C51 language, int, long, float, double type is in opposite order to the high level language. For example: Single Precision Floating Point number 1.0 in high level language presents as: 0x00, 0x00, 0x80, 0x3F. In C51, it is in opposite order : 0x3F， 0x80， 0x00， 0x00.

The DOUBLE type is stored as 8 byte array with the reverse order to PC high level language. For example, doubleprecision floatingpoint 1.0 in PC high level language represents: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x3F, but in C51: 0x3F, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

6.10.1 dbltof

BYTE _dbltof(float *result, DOUBLE x)

Objective：

Convert double-precision floating-point to Single Precision Floating Point number

Parameters：

*result	[out]The address of Single Precision Floating Point number
x	[in]Double-precision floating-point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
            BYTE xdata bRes = 0;
            float xdata fRes;
            DOUBLE xdata dx;

            //dx 为 1.0 ,( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x3F )
        DOUBLE_INIT(dx, 1.0);

            bRes = _dbltof(&fRes, dx);
            if(bRes != 0)
            {
                _set_response(1,&bRes);
                _exit();
            }

            _set_response(4, &fRes); //output: 0x3F 0x80 0x00 0x00
            _exit();
```

6.10.2 ftodbl

BYTE _ftodbl(DOUBLE result, float x)

Objective:

Convert Single Precision Floating Point number to double-precision floating-point

Parameters:

result	[out]Double-precision floating-point number
x	[in]Single Precision Floating Point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.10.3 dbltol

BYTE _dbltol(long *result, DOUBLE x)

Objective:

Convert double-precision floating-point to 32-bit signed int

Parameters:

*result	[out]The address of 32-bit signed int
x	[in]Double-precision floating-point

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.10.4 ltodbl

BYTE _ltodbl(DOUBLE result, long x)

Objective:

Convert 32-bit signed int to double-precision floating-point

Parameters:

result	[out]Double-precision floating-point
x	[in]32-bit signed integer

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.10.5 dtof

BYTE _dtof(float* presult, DOUBLE px)

Objective:

Convert double-precision floating-point to Single Precision Floating Point number

Parameters:

*presult	[out] Single Precision Floating Point number
px	[in]Double-precision floating-point

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

Example:

```
        BYTE xdata bRes = 0;
        float xdata fRes;
        DOUBLE xdata dx;

        //dx is 1.0 ,( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x3F )
    DOUBLE_INIT(dx, 1.0);

        bRes = _dtof(&fRes, dx);
        if(bRes != 0)
        {
            _set_response(1,&bRes);
            _exit();
        }

        _set_response(4, &fRes);
        _exit();
```

6.10.6 ftod

BYTE _ftod(DOUBLE presult, float* px)

Objective:

Convert Single Precision Floating Point number to double-precision floating-point

Input parameters:

presult	[out]Double-precision floating-point
*px	[in] Single Precision Floating Point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.10.7 dtol

BYTE _dtol(long* presult, DOUBLE px)

Objective:

Convert double-precision floating-point number to 32 bit integer.

Parameters:

*presult	[out] 32 bit integer
px	[in] Double-precision floating-point number

Return value:

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.10.8 ltod

BYTE _ltod (DOUBLE presult, long* px)

Objective：

Convert 32 bit integer to double-precision floating-point number

Parameters:

presult	[out]double-precision floating-point number
*px	[in] 32 bit integer

Return value：

Success returns 0. Other return means failure. For more detail, check the error list in section 6.12.

6.11 Structures and Constant Variables

To program in C51, some structures, constant and macro. All of them can be found in RY5_C51.h

6.11.1 Data Type

Some declarations about data type for C51 programming:

```
typedef unsigned char   BOOL;    // Boolean type
typedef unsigned short  word;    // unsigned short type, 16 bit/ 2 byte
typedef unsigned short  WORD;    // unsigned short type, 16 bit/ 2 byte
typedef unsigned long   dword;   // unsigned long type, 32 bit/ 4 byte
typedef unsigned long   DWORD;  // unsigned long type, 32 bit/ 4 byte
typedef unsigned char   byte;      // unsigned character, 1 byte
typedef unsigned char   BYTE;      //unsigned character, 1 byte
typedef BYTE            DOUBLE[8];    // double-precision floating-point , 8 byte array
typedef BYTE            HANDLE; // handle, 1 byte.
```

6.11.2 Macro

```
#define SHA_DIGEST_LENGTH       20    // the length of SHA1 hash value
#define MD5_DIGEST_LENGTH       16    // the length of MD5 hash value
#define DES_KEY_LENGTH          8     // the length of DES key value
#define TDES_KEY_LENGTH         16    // the length of 3DES key value


#define ROCKEY5_SHA_CBLOCK            64    // the length of SHA1 hash block
#define ROCKEY5_MD5_CBLOCK            64    // the length of MD5 hash block


#define RSA_CRYPT_NOPKCS        RSA_CALC_NORMAL // Encryption
#define RSA_CRYPT_PKCS          RSA_CALC_PKCS     // encrypt in #PKCS1 standard
#define RSA_SIGN_NOPH    RSA_CALC_NORMAL // sign the hash value
#define RSA_SIGN_HASH    RSA_CALC_HASH      // Use SHA1 to hash and then sign the hash value
#define RSA_SIGN_PKCS    RSA_CALC_PKCS      // sign the hash value in #PKCS1 standard
// Use SHA1 to hash in #PKCS1 stanard and then sign the hash value
#define RSA_SIGN_PH        (RSA_CALC_HASH|RSA_CALC_PKCS)


#define RSA_VERI_NOPKCS    RSA_CALC_NORMAL      // verify the signature
#define RSA_VERI_PKCS RSA_CALC_PKCS      // verify the signature in #PKCS1 standard


#define RSA_CALC_BIT_512   0x10       //512 bit RSA encryption
#define RSA_CALC_BIT_1024 0x00       // 1024 bit RSA encryption
#define RSA_CALC_BIT_2048 0x20       //2048 bit RSA encryption
```

```
// Convert double-precision floating-point to Single Precision Floating Point number
#define _dtof(result,x)    _dbltof(result,x)
// Convert double-precision floating-point to 32 bit signed integer
#define _dtol(result,x)    _dbltol(result,x)
// Convert Single Precision Floating Point number to double-precision floating-point
#define _ftod(result,x)    _ftodbl(result,*(x))
//Convert 32 bit signed integer to double-precision floating-point
#define _ltod(result,x)    _ltodbl(result,*(x))

// short int (short, int) reversing
#define _swap_u16(pvData)   _swap(pvData,2)
// long type and float type reversing
#define _swap_u32(pvData)   _swap(pvData,4)
// assign Single Precision Floating Point number y to double-precision floating-point x
#define DOUBLE_INIT(x,y) _ftodbl(x,y)
```

6.11.3 Structures and Enumerations

- SHA1 context structure

```
typedef struct _tagSHA_CONTEXT {
        DWORD   h[5];
        DWORD   dwTotalLength;
        BYTEbRemainLength;
        BYTEpbRemainBuf[ROCKEY5_SHA_CBLOCK];
}SHA_CONTEXT,*PSHA_CONTEXT;

typedef SHA_CONTEXT ShaContext;             //SHA_CONTEXT structure type
typedef ShaContext* PShaContext;       // The pointer pointing to SHA_CONTEXT
```

- MD5 hash context structure

```
typedef struct _tagMD5_CONTEXT {
    DWORD h[4];
    DWORD dwTotalLength;
    BYTE    bRemainLength;
    BYTE    pbRemainBuf[ROCKEY5_MD5_CBLOCK];
}MD5_CONTEXT,*PMD5_CONTEXT;

typedef MD5_CONTEXT Md5Context;         //MD5_CONTEXT structure type
typedef Md5Context* PMd5Context;    //The pointer pointing to MD5_CONTEXT
```

- File structure

```
typedef struct _FILE_INFO {
    WORD            wFileID;    //File ID
    BYTE        bFileType;      //File type
    WORD            wFileSize;//File size
}EFINFO,*PEFINFO;
```

- File type

```
enum
{
    FILE_TYPE_EXE = 0x00,   //executable file
    FILE_TYPE_DATA,                 //data file
    FILE_TYPE_RSA_PUBLIC, //RSA public key file
    FILE_TYPE_RSA_PRIVATE,      //RSA private key file
};
```

- Create file

```
enum
```

```
{
        //if the file is already existed, open the file. Otherwise create it and then open
        CREATE_OPEN_ALWAYS = 0x00,
        //create and open the new file, if the file is existed then return error.
        CREATE_FILE_NEW,
        // open existed file as open()
        CREATE_OPEN_EXISTING
};
```

- Hardware information

```
enum
{
        GLOBAL_SERIAL_NUMBER = 0x00,        // get 8 byte series
        GLOBAL_CLIENT_NUMBER,               // get 4 byte vendor ID
        GLOBAL_COS_VERSION                  //get 2 byte COS version
};
```

- Encryption/Decryption and signature

```
enum
{
        RSA_CALC_NORMAL     = 0x00,    // no encoding and just calculate
        RSA_CALC_HASH,      // Hash the imported value and encrypt by private key
        RSA_CALC_PKCS,      // encrypt by PKCS#1 standard
};
```

6.12 Error Code

Error code:

```
#define    RY5_C51_SUCCESS                0x00000000    //Success
#define    RY5_C51_UNKNOWN                0x00000001    //unknow error
```

```
#define    RY5_C51_INVALID_PARAMETER         0x00000002    //invalid parameter
#define    RY5_C51_INVALID_ADDRESS           0x00000003    // invalid address
#define    RY5_C51_INVALID_SIZE              0x00000004    // invalid size
#define    RY5_C51_FILE_NOT_FOUND            0x00000005    //file not found
#define    RY5_C51_ACCESS_DENIED             0x00000006    // Access file denied
#define    RY5_C51_FILE_SELECT           0x00000007    // open file number has reached its maximum
#define    RY5_C51_INVALID_HANDLE        0x00000008    // invalid handle
#define    RY5_C51_FILE_OUT_OF_RANGE         0x00000009    // file out of range
#define    RY5_C51_FILE_TYPE_MISMATCH    0x0000000A    // file type mismatch
#define    RY5_C51_FILE_SIZE_MISMATCH    0x0000000B    // file size mismatch
#define    RY5_C51_NO_SPACE                  0x0000000C    //have no enough space
#define    RY5_C51_FILE_EXIST            0x0000000D    //the file or its child directory is already existed.
#define    RY5_C51_INVALID_KEY_FORMAT    0x0000000E    // invald RSA key format
#define    RY5_C51_KEY_LEN_MISMATCH       0x0000000F    //mismatched the length of key
#define    RY5_C51_RSA_INVALID_KEY_FILE      0x00000010    // file type invalid
#define    RY5_C51_RSA_ENC_DEC_FAILED    0x00000011    // failed in RSA encryption or decryption
#define    RY5_C51_RSA_SIGN_VERI_FAILED      0x00000012    // failed in RSA verification
#define    RY5_C51_SHA1                      0x00000013    // SHA1 computing error
#define    RY5_C51_MD5                       0x00000014    // MD5 computing error
#define    RY5_C51_INVALID_ADDERSS           0x00000015    // invalid memory pointer
#define    RY5_C51_EEPROM                    0x00000016    // EEPROM error
#define    RY5_C51_FUNC_EXPIRED              0x00000017    //COS expired
#define    RY5_C51_HW_CLOCK_BROKEN           0x00000018    //hardware clock broken
#define    RY5_MALTCIOUS_UPDATE              0x00000019   // update package expired
```

Chapter 7. ROCKEY5 TIME API (For PC)

7.1 API Functions

7.1.1 RY5_Find

DWORD WINAPI RY5_Find(char* pVendorID, int* pCount);

Objective:

Search the connected ROCKEY 5 TIME device

Parameters:

* pVendorID	[in/out] String pointer pointing to the string of vendor ID and the vendor ID length is 8 bytes.
* pCount	[out] Int pointer to the number of attached ROCKEY5 TIME

Note:

■The function gets the device number of ROCKEY5 TIME for certain vendor ID, if * pVendorID is assigned as 0, then * pCount return 0

■The maximum number of ROCKEY 5 TIME attached to PC is 32 at the same time

■The function is at least needed to be called once for initalzation and setup the device list when the program gets started.

■If the program is multi-thread, it is better to call this function once before the main thread gets started.

■The attaching and detaching of dongle will not affect the number of device.

Return value：

Success returns RY5_SUCCESS（0x00000000）. Other return means failure. For more detail, check the error list in section 7.2.

Example:

Check RY5_close example

7.1.2 RY5_Open

DWORD WINAPI RY5_Open(RY_HANDLE* pHandle, int Item);

Objective:

Open ROCKEY 5 TIME device. RY5_Find must be called before calling this function.

Parameters:

* pHandle	[out] Handle pointer. Return opened ROCKEY5 TIME handle
Item	[in]Open the specified ROCKEY5 TIME, starting from 1.

Note:

■Open successfully. * pHandle points to the opened ROCKEY5 TIME device handle

■The function opens in sharing mode, allowing other process access the ROCKEY5 TIME. In this mode, it supports multi-process. High frequency of open and close operation can reduce the program performance.

■For one program opens multiple ROCKEY5 TIME, one should define handle array to store opened handle for next step. Or operate each opened ROCKEY5 TIME in a loop.

■The user should call RY5_Close to release the handle after finishing the operation.

Return value：

Success returns RY5_SUCCESS（0x00000000）. Other return means failure. For more detail, check 7.2.1 error lists.

Example:

Check RY5_Close example

7.1.3 RY5_Close

DWORD WINAPI RY5_Close(RY_HANDLE handle , BOOL IsReset);

Objective:

Close ROCKEY5 TIME device. RY5_Open should be called before calling RY5_Close.

Parameters:

handle	[in] Handle to ROCKEY5 TIME dongle to be closed.
IsReset	[in] The safe status flag. If set true, the safe status and data in ROCKEY5 TIME memory will be cleaned. Else if set false, the safe status and data will be kept.

Note:

The function is to close the ROCKEY5 TIME. If IsReset is true, it clears up the device memory. Else IsReset is false, it keeps the information.

Return value：

Success returns RY5_SUCCESS（0x00000000）. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
           RY_HANDLE  handle = 0;     // the device handle
           DWORD retcode;            // the error code
           int   count = 0;          // the dongle number
           char  vendorID[10]="A8C3E6FD";  // vendor ID
           int i;

           retcode = RY5_Find(vendorID, &count);   // find the certain vendor ID's
dongle.

           if ( retcode != RY5_SUCCESS)
           {
               printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
               return;
           }

           printf("\r\n RY5_Find()success，the found dongle number：%d\r\n", count);


           for(i=1; i<=count; i++)
           {
               retcode = RY5_Open(&handle, i);  // open ith dongle
               if ( retcode != RY5_SUCCESS)
               {
                   printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
                   return;
               }

               printf("\r\n RY5_Open()success，open the %d dongle\r\n", i);

               //
               // the other operation of dongle can be written here
               //

               retcode = RY5_Close(handle, TRUE);  // close dongle and clear safe status
               if ( retcode != RY5_SUCCESS)
               {
                   printf("\r\n RY5_Close()error，the error code：%08X \r\n", retcode);
                   return;
               }

               printf("\r\n RY5_Close() suceess，close No. %d dongle\r\n", i);

           }
```

7.1.4 RY5_GetHardID

DWORD WINAPI RY5_GetHardID(RY_HANDLE handle, char* pbuf);

Objective:

Get the serial number of ROCKEY5 TIME dongle. RY5_Open must be called before the function is called

Parameters:

handle [in] The device handle which is as same as RY5_Open return value.

*pbuf [out]The pointer of hardware serial number

Note:

The return hardware serial number is a 16- byte string.

Return value：

Success returns RY5_SUCCESS（0x00000000）. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      // the device handle
DWORD retcode;            // the error code
int count = 0;            // the count of found dongle
char vendorID[10]="A8C3E6FD";   // vendor ID
char buf[20];   // to store the return hardware ID
int i;

//find
retcode = RY5_Find(vendorID, &count); //find the dongle of the certain
                                        //Vendor ID
if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error， error code：%08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find()suceess， find dongle count：%d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open dongle ith
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error， error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open()success， Open %dth dongle\r\n", i);

    //Get HardID
    memset(buf, 0, sizeof(buf));
    retcode = RY5_GetHardID(handle, buf);  // get the hardware ID
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_GetHardID()error，error code：%08X \r\n", retcode);
        return;
    }
```

```
            printf("\r\n RY5_GetHardID()success， hardware ID：%s\r\n",buf);

            //close
            retcode = RY5_Close(handle, TRUE);  // close dongle and clear the flag
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Close()error， error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Close()success，close %dth dongle\r\n", i);
        }
```

7.1.5 RY5_GenRandom

DWORD WINAPI RY5_GenRandom(RY_HANDLE handle, int len_need, BYTE* pOutbuf);

Objective:

Get the specified length random number. RY5_Open must be called before using this function

Parameters:

handle	[in]The device handle which is as same as the one returned by RY5_Open
len_need	[in] The length of random number
*pOutbuf	[out]The random number pointer

Note:

The maximum length of random number is 16 bytes. If the length is longer than 16 bytes, the system only returns 16 bytes random number without error report.

Return value：

Success returns RY5_SUCCESS（0x00000000）. Other return means failure. For more detail, check section 7.2.

Example:

```
        RY_HANDLE handle = 0;      // the device handle
        DWORD retcode;             // error code
        int count = 0;             // the dongle count
        char vendorID[10]="A8C3E6FD";   // the certain vendor ID
        BYTE buf[18];  // the return random number
        int i;

        //find
        retcode = RY5_Find(vendorID, &count);   // search the certain vendor ID
                                                  //dongle
        if ( retcode != RY5_SUCCESS)
        {
```

```
        printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Find()success，search the dongle count：%d\r\n", count);

    for(i=1; i<=count; i++)
    {
        //open
        retcode = RY5_Open(&handle, i);  // open dongle ith
        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Open() error， error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Open()success， open dongle \r\n", i);

        //Get Random
        memset(buf, 0, sizeof(buf));
        retcode = RY5_GenRandom(handle,10, buf);  // get the 10 byte random number
        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_GenRandom() error，error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_GenRandom()success， random number：%08x\r\n",buf);

        //close
        retcode = RY5_Close(handle, TRUE);  // close dongle and clear safe flag
        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Close() error， error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Close()success， close dongle \r\n", i);
    }
```

7.1.6 RY5_GetFreeSize

DWORD WINAPI RY5_GetFreeSize(RY_HANDLE handle, int* pSize);

Objective:

Get the free user space. RY5_Open must be called to open the device before this function is called.

Parameters:

handle	[in]The device handle which is as same as the one returned by RY5_Open

*pSize [out]The pointer pointing to the free user space

Note:

Check the free user space of dongle.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
            RY_HANDLE handle = 0;      // the device handle
            DWORD retcode;             // error code
            int count = 0;             // search the dongle number
            char vendorID[10]="A8C3E6FD";   // find the vendor ID
            int pSize;  // return the size of the free space
            int i;

            //find
            retcode = RY5_Find(vendorID, &count);   // find the certain vendor ID

            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Find() error， error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Find()success， find the dongle count：%d\r\n", count);

            for(i=1; i<=count; i++)
            {
                //open
                retcode = RY5_Open(&handle, i);  // open dongle
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_Open() error， error code：%08X \r\n", retcode);
                    return;
                }

                printf("\r\n RY5_Open()success，open the dongle\r\n", i);

                //Get FreeSize
                retcode = RY5_GetFreeSize(handle,&pSize);  // get the free space size
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_GetFreeSize() error， error code：%08X \r\n",
retcode);
                    return;
                }
```

```
              printf("\r\n RY5_GetFreeSize() success， free space：%d byte
\r\n",pSize);

              //close
              retcode = RY5_Close(handle, TRUE);  // close dongle and clear the safe
                                                  //flag
              if ( retcode != RY5_SUCCESS)
              {
                  printf("\r\n RY5_Close() error， error code：%08X \r\n", retcode);
                  return;
              }

              printf("\r\n RY5_Close() success， close dongle\r\n", i);

          }
```

7.1.7 RY5_LEDControl

DWORD WINAPI RY5_LEDControl(RY_HANDLE handle, int flag);

Objective:

LED controller.   RY5_Open must be called to open the device before this function is called.

Parameters:

handle [in] The device handle which is as same as the one returned by RY5_Open

flag [in]Instruction. Please refer to the following for more information

Note:

LED definition:

#define ET_LED_ON 1 // LED on

#define ET_LED_OFF 2 // LED off

#define ET_LED_WINK 3 // LED twinkle

Return value：

Success returns RY5_SUCCESS（0x00000000）. Other return means failure.   For more detail, check the error list in section 7.2.

Example:

```
           RY_HANDLE handle = 0;      // device handle
           DWORD retcode;            // error code
           int count = 0;            // the found dongle count
           char vendorID[10]="A8C3E6FD";   // the vendor ID
           int i;

           //find
```

```
retcode = RY5_Find(vendorID, &count);  //find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find() error , error code : %08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() success , the dongle count : %d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open dongle i
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open() error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() success , Open dongle\r\n", i);

    //Control LED
    retcode = RY5_LEDControl(handle,ET_LED_WINK);  // LED twinkle
    if ( retcode != RY5_SUCCESS)
    {
    printf("\r\n RY5_LEDControl() error , error code : %08X \r\n", retcode);
    return;
    }

    printf("\r\n RY5_LEDControl() success , LED twinkle\r\n");

    printf("Please press any key to continue : ");
    getchar();
    retcode = RY5_LEDControl(handle,ET_LED_OFF);  // LED off
    if ( retcode != RY5_SUCCESS)
    {
    printf("\r\n RY5_LEDControl() error , error code : %08X \r\n", retcode);
    return;
    }

    printf("\r\n RY5_LEDControl() success , LED off\r\n");

    printf("Please press any key to continue : ");
    getchar();
    retcode = RY5_LEDControl(handle,ET_LED_ON);  // LED on
    if ( retcode != RY5_SUCCESS)
    {
    printf("\r\n RY5_LEDControl() error , error code : %08X \r\n", retcode);
    return;
```

```
            }

            printf("\r\n RY5_LEDControl() success , LED on\r\n");

            //close
            retcode = RY5_Close(handle, TRUE);  //close dongle and clear the safe flag
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Close() error , error code : %08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Close() success , close the dongle\r\n", i);
        }
```

7.1.8 RY5_VerifyDevPin

DWORD WINAPI RY5_VerifyDevPin(RY_HANDLE handle, char* pInPin, int* pRemainCount);

Objective:

Verify the developer password. RY5_Open must be called to open the device before this function is called.

Parameters:

handle	[in]The device handle which is as same as the one returned by RY5_Open
*pInPin	[in]The pointer points to the string of developer password
*pRemainCount	[out]The pointer points to retry times of developer password

Note:

- The developer pin has not retry limits by default. If developersets set the retry times, the dongle can be locked when reaching retry times. The default developer password is “123456781234567812345678”.
- If it is successful called, *pRemainCount returns 0.
- Returned error code: 0xF0000CXX means developer password error and the last two bits XX is the retry limits. The return value of *pRemainCount is the developer password retry times: 0: locked; 1-254: the remain time; 255: no limits
- When the function return other error code, *pRemainCount returns 0
- The length of developer password must be 24 bytes

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return value means failure. For more detail, check the error list in section 7.2.

Example:

```
            RY_HANDLE handle = 0;      // device handle
            DWORD retcode;            // error code
            int count = 0;            // the dongle count
            char vendorID[10]="A8C3E6FD";   // the vendor ID
            char pInPin[30]="123456781234567812345678"; // the developer password
            int pRemainCount=0;  // the retry times of the developer password
            int i;

            //find
            retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Find() error， error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Find() success， the dongle count：%d\r\n", count);

            for(i=1; i<=count; i++)
            {
                //open
                retcode = RY5_Open(&handle, i);  // Open the dongle whose number is ‘i’
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_Open() error， error code：%08X \r\n", retcode);
                    return;
                }

                printf("\r\n RY5_Open() success， open dongle\r\n", i);

                //Verify DevPin
                retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);//verify
                                                             // developer ID
                if ( retcode != RY5_SUCCESS)
                {
                printf("\r\n RY5_VerifyDevPin() error， error code：%08X
pRemainCount=%d\r\n ", retcode,pRemainCount);
                    return;
                }

                printf("\r\n RY5_VerifyDevPin()  success\r\n");

                //close
                retcode = RY5_Close(handle, TRUE);  //close dongle and clear the safe flag
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_Close() error， error code：%08X \r\n", retcode);
                    return;
                }

                printf("\r\n RY5_Close() success， close dongle\r\n", i);
            }
```

7.1.9 RY5_SetVendorID

DWORD WINAPI RY5_SetVendorID(RY_HANDLE handle, char* pSeed, int len, char* pOutVendorID);

Objective:

Set vendor ID. RY5_Open must be called to open the device and the developer password needs to be verified before this function is called.

Parameters:

handle	[in]The device handle, return value is the same as the one returned by RY5_Open
*pSeed	[in]The pointer points t o the seed
len	[in]The length of seed
*pOutVendorID	[out]The pointer points to vendor ID

Note:

- The vendor ID from user is a string with the maximum length of 250 bytes,
- The developer ID must be verified before this function is called

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      // the device handle
DWORD retcode;             //  error code
int count = 0;             // dongle count
char vendorID[10]="A8C3E6FD";   // vendor ID
char pInPin[30]="123456781234567812345678"; // the developer password
int pRemainCount=0;  // the retry times of developer password
int i;

//find
retcode = RY5_Find(vendorID, &count);  //find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS
{
    printf("\r\n RY5_Find() error , error code：%08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() success , find the dongle count：%d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open the dongle
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open() error， error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() ， \r\n", i);

    //Verify DevPin
   retcode=RY5_VerifyDevPin(handle,pInPin,&pRemainCount);
                                                //verify   developer pwd
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_VerifyDevPin() error， error code：%08X ,
              pRemainCount=%d\r\n ", retcode,pRemainCount);
        return;
    }

    printf("\r\n RY5_VerifyDevPin() success\r\n");

    //set vendorID
    char seed[10]="12345"; // seed
    char buf[10]; //store 8 byte vendor ID
    memset(buf, 0, sizeof(buf));
    //set vendor ID
    retcode = RY5_SetVendorID(handle, seed, lstrlen(seed), buf);
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_SetVendorID() error， error code：%08X \r\n",
             retcode);
        return;
    }

    printf("\r\n RY5_SetVendorID() success，vendor ID：%s\r\n",buf);

    //close
retcode = RY5_Close(handle, TRUE);//close the dongle and clear the safe flag
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Close() error， error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Close() success， close dongle\r\n", i);
}
```

7.1.10 RY5_ChangeDevPin

DWORD WINAPI RY5_ChangeDevPin(RY_HANDLE handle, char* pOldPin, char* pNewPin, int TryCount);

Objective:

Reset vendor ID to get the regarding permission. RY5_Open must be called to open the device before this function is called.

Parameters:

handle	[in]Device handle which is the same as the one returned by RY5_Open.
*pOldPin	[in]The pointer points to the old developer password
*pNewPin	[in]The pointer points to the new pin
TryCount	[in]The retry count

Note:

- TryCount is the retry times ranging from 1 to 255. 255 means no limits
- Reset the pin interface includes the verification of developer password function regarding to password lock. Please check RY5_VerifyDevPin for more detail.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
        RY_HANDLE handle = 0;      // device handle
        DWORD retcode;            // error code
        int count = 0;            // the dongle count
        char vendorID[10]="A8C3E6FD";   // vendor ID
        int i;

        //find
        retcode = RY5_Find(vendorID, &count);   //  find vendor ID

        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Find() error， error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Find() success， find dongle：%d\r\n", count);

        for(i=1; i<=count; i++)
        {
            //open
            retcode = RY5_Open(&handle, i);  // open dongle
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Open() error， error code：%08X \r\n", retcode);
                return;
            }
            printf("\r\n RY5_Open() success， open dongle\r\n", i);

            //Change DevPin
            char OldPin[30]="123456781234567812345678"; // developer pin
            char NewPin[30]="111111111111111111111111"; // new pin
           //Reset the developer pin and no retry limits
            retcode = RY5_ChangeDevPin(handle, OldPin, NewPin, 255);
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_ChangeDevPin() error， error code：%08X \r\n",
                      retcode);
                return;
            }

            printf("\r\n RY5_ChangeDevPin() success\r\n");

            //close
            retcode = RY5_Close(handle, TRUE);  // close dongle and clear safe flag
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Close() error， error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Close() success， close dongle\r\n", i);
        }
```

7.1.11 RY5_Read

DWORD WINAPI RY5_Read(RY_HANDLE handle, int offset, BYTE* pOutbuf, int len);

Objective:

Read the content in data area of dongle. RY5_Open must be called to open the device before this function is called.

Parameters:

handle	[in]The device handle which is the same as the one returned by RY5_Open
offset	[in]Offset address where reading operation starts from
*pOutbuf	[out] The buffer pointer
len	[in]Read length

Note:

Memory space is 8 KB. 0-7178 bytes are in lower zone. 7168-8192 bytes are in upper zone.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      // device handle
DWORD retcode;            // error code
int count = 0;            // the dongle count
char vendorID[10]="A8C3E6FD";   // the vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count);   // search certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find() error， error code：%08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() success， the dongle count：%d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open dongle i
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open() error， error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() success， open dongle\r\n", i);

    //Read Memery
    int offset = 0;  // offset address
    int len = 10;   // read length
    BYTE buf[30];  // store the data
    memset(buf, 0, sizeof(buf));
    retcode = RY5_Read(handle, offset, buf, len);// read the data area content
                                        // offset address 0, read length 10
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Read() error， error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Read() success，read content：%08X\r\n",buf);

    //close
    retcode = RY5_Close(handle, TRUE);  // close dongle and clear safe flag
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Close() error， error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Close() success， close dongle\r\n", i);
}
```

7.1.12 RY5_Write

DWORD WINAPI RY5_Write(RY_HANDLE handle, int offset, BYTE* pInbuf, int len);

Objective:

Write data to the low data area of dongle. RY5_Open must be called to open the device before this function is called.

Parameters:

handle	[in]Device handle which is the same as the handle returned by RY5_Open
offset	[in]Offset address
*pInbuf	[in]The pointer points to the buffer for storing the content to be written.
len	[in]The written data length

Note:

The data area is 8192 byte. 0-7168 bytes are in the lower zone; 7169-8192 bytes are in upper zone. This function can only be called to write data in lower zone (0-7168).

Return value：

Success returns RY5_SUCCESS（0x00000000）. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      // device handle
DWORD retcode;            // error code
int count = 0;            // the dongle count
char vendorID[10]="A8C3E6FD";   // the certain vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count);   //  find the certain vendor ID

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find() error， error code：%08X \r\n", retcode);
    return;
}
```

```
        printf("\r\n RY5_Find() success， find the dongle count：%d\r\n", count);

        for(i=1; i<=count; i++)
        {
            //open
            retcode = RY5_Open(&handle, i);  // open dongle
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Open() error， error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Open() success， open dongle\r\n", i);

            //Write Memery
            int offset = 0;  // offset address
            int len = 5;   // the readin data length
            BYTE buf[10]="hello";  // data buffer
            retcode = RY5_Write(handle, offset, buf, len); //write data into data
                                                       // area of dongle
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Write() error， error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Write() success， write content：%s\r\n",buf);

            //close
            retcode = RY5_Close(handle, TRUE);   //close dongle and clear the safe flag
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Close() error， error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Close() success， close the dongle\r\n", i);
        }
```

7.1.13 RY5_VendorWrite

DWORD WINAPI RY5_VendorWrite(RY_HANDLE handle, int offset, BYTE* pInbuf, int len);

Objective:

The developer writes data to the data area. The function is only used by developer.

RY5_Open must be called to open the device and the developer password needed to be verified before this function is called.

Parameters:

handle [inDevice handle which is the same as the one returned by RY5_Open

offset [inThe offset address

*pInbuf [in]The pointer points to the buffer for storing data to be written

len [in]Write length

Note:

■The data area is 8192 byte. 0-7168 byte is the low level store area; 7169-8192 byte is high level store area.

■This function can write data to all data area. Only developer can call this function and RY5_VerifyDevPin needs to be called before calling this function.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handledevice handle
DWORD retcode;             // error code
int count = 0;             // the dongle count
char vendorID[10]="A8C3E6FD";   // vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count);  // find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find() error , error code : %08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() success , find dongle count : %d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open dongle
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open()  , \r\n", i);

    //Verify DevPin
    char pInPin[30]="111111111111111111111111"; //developer password
    int pRemainCount=0;  //pin remain count
    //verify developer password
    retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_VerifyDevPin()error , error code : %08X
        pRemainCount=%d\r\n ", retcode,pRemainCount);
        return;
    }

    printf("\r\n RY5_VerifyDevPin() success , verify developer pin
         success\r\n");

    //Write Memery
    int offset = 0;  //offset address
    int len = 5;   // data length
    BYTE buf[10]="hello";  // data buffer
    //write data to the data area of dongle
    retcode = RY5_VendorWrite(handle, offset, buf, len);
    if ( retcode != RY5_SUCCESS)
    {
     printf("\r\n RY5_VendorWrite()error , error code : %08X \r\n", retcode);
     return;
    }

    printf("\r\n RY5_VendorWrite() success ,  write : %s\r\n",buf);

    //close
retcode = RY5_Close(handle, TRUE);//close the dongle and clear the safe flag
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Close()error , error code : %08X  \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Close() success ,  close dongle\r\n", i);
}
```

7.1.14 RY5_ReadShare

DWORD WINAPI RY5_ReadShare(RY_HANDLE handle, int offset, BYTE* pbuf, int len);

Objective:

Read the share memory area of dongle. RY5_Open must be called to open the device before this function is called.

Parameters:

handle	[in]Device handle, as same as the handle returned from RY5_Open.

offset [in] offset address. The position to read data from.

*pbuf [out]The pointer points to the buffer for storing the read content.

len [in]Read length.

Note:

The size of the shared memory is 32 bytes.

Return value：

Success returns RY5_SUCCESS（0x00000000）. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handle
DWORD retcode;            //error code
int count = 0;            //the found dongle count
char vendorID[10]="A8C3E6FD";   //vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count); //search the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() ，the found dongle count：%d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open dongle i
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() success， open dongle\r\n", i);

    //Read Share
    int offset = 0;  //offset address
    int len = 10;   // read length
    BYTE buf[30];  // the data buffer
    memset(buf, 0, sizeof(buf));
   // read the data in shared area
    retcode = RY5_ReadShare(handle, offset, buf, len);
    if ( retcode != RY5_SUCCESS)
```

```
            {
                printf("\r\n RY5_ReadShare()error , error code : %08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_ReadShare() ,: %08X\r\n",buf);

            //close
        retcode = RY5_Close(handle, TRUE);//close the dongle and clear the safe flag
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Close()error , error code : %08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Close()  ,  \r\n", i);
        }
```

7.1.15 RY5_WriteShare

DWORD WINAPI RY5_WriteShare(RY_HANDLE handle, int offset, BYTE* pbuf, int len);

Objective:

Write the data to the shared memory of dongle. RY5_Open must be called to open the device before this function is called.

Parameters:

handle	[in]Device handle, as same as the handle returned from RY5_Open
offset	[in] offset address. The position writting data from.
*pbuf	[in]The pointer points to the buffer for storing the data to be written.
len	[in] The length of data to be written.

Note:

The size of shared memory is 32 bytes. The user can operate the shared area to prevent multiple clients from accessing dongle and prevent USB data communication simulation.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
        RY_HANDLE handle = 0;      //device handle
        DWORD retcode;             //error code
        int count = 0;             //the found dongle count
        char vendorID[10]="A8C3E6FD";   //vendor ID
```

```
int i;

//find
retcode = RY5_Find(vendorID, &count);  //find the certain vendor ID’s dongle.

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error , error code : %08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() , the found dongle count : %d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open dongle.
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() success ,  open dongle \r\n", i);

    //Write Share
    int offset = 0;  //offset address
    int len = 5;   // the length of data to be written
    BYTE buf[10]="hello";  //data to be written
    retcode = RY5_WriteShare(handle, offset, buf, len);

    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_WriteShare()error ,error code :%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_WriteShare() was called successfully! Written
          data  : %s\r\n",buf);

    //close
retcode = RY5_Close(handle, TRUE); //close the dongle and clear the safe flag
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Close()error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Close() was called successfully , No. %d dongle was
     closed \r\n", i);
}
```

7.1.16 RY5_CreateFile

DWORD WINAPI RY5_CreateFile(RY_HANDLE handle, WORD FileID, int Size, int Type);

Objective:

Create file. RY5_Open must be called to open the device and the developer password needs to be verified before this function is called.

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open。
FileID	[in]File ID
Size	[out]File size
Type	[in]File type

Note:

■The free user space of ROCKEY5 TIME is 32k

■File ID is 2 bytes. For example, 0x1002 and 0x009B. Some IDs can only be used internally in ROCKEY5 TIME. For more details, please refer to appendix.

■The file size(not including the file property information 16 bytes) needs to be defined when creating file. For example, creating a 100 bytes file (size =100), the actual obtained space is 100+16= 116 bytes.

■Type is used to set file type. Please check the table below.

FILE_TYPE_EXE	The executable file. It can be written once developer password is verified or from other executable file in ROCKEY5 TIME. This file cannot be read in any circumstances.
FILE_TYPE_DATA	Internal data file. Write the data by executable file in ROCKEY5 TIME. The external operation can delete and write but not rea after the verification of developer password.
FILE_TYPE_RSA_PUBLIC	RSA public key file. The public key file can be written after the verification of developer password or just from the executable file in ROCKEY5 TIME.

FILE_TYPE_RSA_PRIVATE	RSA private key file. The private key file can be written after the verification of developer password or just from the executable file in ROCKEY5 TIME. No way to read private key.

Table 7-1 File type

File type definition:

```
#define FILE_TYPE_EXE            0    // executable file
#define FILE_TYPE_DATA           1    // data file
#define FILE_TYPE_RSA_PUBLIC     2    //RSA public key file
#define FILE_TYPE_RSA_PRIVATE    3    //RSA private key file
```

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
        RY_HANDLE handle = 0;      // device handle
        DWORD retcode;            //error code
        int count = 0;            //the found dongle count
        char vendorID[10]="A8C3E6FD";   //vendor ID
        int i;

        //find
        retcode = RY5_Find(vendorID, &count);  // find the certain vendor ID’s dongle

        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Find()error , error code : %08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Find() , the found dongle count : %d\r\n", count);

        for(i=1; i<=count; i++)
        {
            //open
            retcode = RY5_Open(&handle, i);  // open the dongle whose No. is ‘i’.
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Open()error , error code : %08X \r\n", retcode);
                return;
            }
```

```
                printf("\r\n RY5_Open() success， open dongle No. %d\r\n", i);

                //Verify DevPin
                char pInPin[30]="123456781234567812345678"; //developer password
                int pRemainCount=0;  //developer password retry times
               //verify developer password
                retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_VerifyDevPin()error，error code：%08X
                    pRemainCount=%d\r\n ", retcode,pRemainCount);
                    return;
                }

                printf("\r\n RY5_VerifyDevPin() success.\r\n");

                //Create File
                WORD fileID = 0x1002;  //offset address
                int fileSize = 100;   // thelength of data to be written
        //To create internal data file in dongle. File ID=1002, length=100 bytes.
                retcode = RY5_CreateFile(handle, fileID, fileSize, FILE_TYPE_DATA);

           if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_CreateFile()error ,error code :%08X \r\n", retcode);
                    return;
                }

                printf("\r\n RY5_CreateFile() success.");

                //close
            retcode = RY5_Close(handle, TRUE);//close the dongle and clear the safe flag
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_Close()error，error code：%08X \r\n", retcode);
                    return;
                }

                printf("\r\n RY5_Close() success， No. %d dongle was closed\r\n", i);
            }
```

7.1.17 RY5_WriteFile

DWORD WINAPI RY5_WriteFile(RY_HANDLE handle, WORD FileID,int offset,BYTE* pbuf,int Size);

Objective:

RY5_Open must be called to open the device and the verification is needed for developer pin before this function is called

Parameters:

handle [in]Device handle, as same as the handle returned by RY5_Open

FileID	[in]File ID
offset	[in]offset address
pbuf	[in]The pointer points to the buffer for storing data to be written
Size	[in]The size of data to be written

Note:

- ■The free user space of ROCKEY5 TIME is 32k
- ■When writing data to the file, ensure that the file is existed. If there is no such file, one needs to create a file(RY5_CreateFile)
- ■File ID is 2 bytes. For example, 0x1002 and 0x009B. For some reason, some IDs can only be used internally in ROCKEY5 TIME. For more details, please refer to appendix.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, please check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handle
DWORD retcode;            //error code
int count = 0;            //the found dongle count
char vendorID[10]="A8C3E6FD";   //vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error , error code : %08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() , the found dongle count : %d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open the dongle whose No. is ‘i’
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() was called successfully, No. %d dongle was
           open\r\n", i);

    //Verify DevPin
    char pInPin[30]="123456781234567812345678"; //developer password
    int pRemainCount=0;  //developer retry times
    //verify developer pin
    retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_VerifyDevPin()error，error code：%08X
         pRemainCount=%d\r\n ", retcode,pRemainCount);
        return;
    }

    printf("\r\n RY5_VerifyDevPin() ，verify developer pin 。\r\n");

    //Create File
    WORD fileID = 0x1006;  //offset address
    int fileSize = 100;   // the length of file
    retcode = RY5_CreateFile(handle, fileID, fileSize, FILE_TYPE_DATA);
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_CreateFile()error ,error code :%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_CreateFile() success，create file successfully");

    //Write File
    int offset = 0;  // offset
    BYTE buf[30]="hello";
    retcode = RY5_WriteFile(handle, fileID, offset, buf, fileSize);
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_WriteFile()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_WriteFile() success，write file successfully");

    //close
retcode = RY5_Close(handle, TRUE); //close the dongle and clear the safe flag
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Close()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Close() success， close dongle successfully\r\n", i);
}
```

7.1.18 RY5_ExecuteFile

DWORD WINAPI RY5_ExecuteFile(RY_HANDLE handle,

WORD FileID,

BYTE* pInBuf,

int InSize,

BYTE* pOutBuf,

int* pOutSize);

Objective:

Execute the certain file. RY5_Open must be called to open the device and the developer password needs to be verified before this function is called

Parameters:

handle [in]Device handle, as same as the handle returned by RY5_Open。

FileID [in]File ID

*pInBuf [in]The pointer points to the buffer for storing input to the C51 executable file.

InSize [in]The data length

*pOutBuf [out]The pointer points to output buffer for storing data output from the C51 executable file

*pOutSize [in/out] The length of data returned from C51 Executable file.

Note:

■File ID is 2 bytes. For example 0x1002、0x009B. For some reason, some IDs can only be used by ROCKEY5 TIME internally. Users cannot use these internal IDs..

■It is the size of the output buffer pointed by pointer *pOutBuf. When it is shorter than the returned length from the executable file from ROCKEY 5 TIME, it returns error.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in 7.2.

Example:

```
        RY_HANDLE handle = 0;      //device handle
        DWORD retcode;             //error code
        int count = 0;             //the found dongle count
        char vendorID[10]="A8C3E6FD";   //the vendor ID
        int i;
```

```
//find
retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() ，the found dongle count：%d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open the dongle whose No. is ‘i’
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() success， open No. %d dongle\r\n", i);

    //Execute File
    WORD fileID = 0x2001;  //file ID
    int inSize = 10;   // input data length
    int outSize = 10; // output data length
    BYTE inBuf[30] = "1";
    BYTE outBuf[30] = "0";
    retcode = RY5_ExecuteFile(handle, fileID, inBuf,
      inSize,outBuf,&outSize); //run the executable file
    if ( retcode != RY5_SUCCESS)
    {
    printf("\r\n RY5_ExecuteFile()error，error code：%08X \r\n", retcode);
    return;
    }

    printf("\r\n RY5_ExecuteFile() success，return result：%s\r\n",outBuf);

    //close
retcode = RY5_Close(handle, TRUE); //close the dongle and clear the safe flag
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Close()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Close() success， the No. %d dongle was closed\r\n", i);
}
```

7.1.19 RY5_EraseAllFile

DWORD WINAPI RY5_EraseAllFile(RY_HANDLE handle);

Objective:

Delete all the files. RY5_Open must be called to open the device and the developer password needed to be verified before this function is called.

Input parameters:

handle [in]Device handle, as same as the handle returned from RY5_Open。

Note:

This function is called to delete all the files.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handle
DWORD retcode;             //error code
int count = 0;             //the found dongle count
char vendorID[10]="A8C3E6FD";   //vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error , error code : %08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() , the found dongle count : %d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open dongle whose No. is ‘i’
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() success , open the No. %d dongle\r\n", i);
```

```
      //Verify DevPin
      char pInPin[30]="123456781234567812345678"; //developer password
      int pRemainCount=0;  //pin remain count
     //verify developer password
      retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);
      if ( retcode != RY5_SUCCESS)
      {
        printf("\r\n RY5_VerifyDevPin()error , error code : %08X ,
        pRemainCount=%d\r\n ", retcode, pRemainCount);
        return;
      }

      printf("\r\n RY5_VerifyDevPin()  , verify developer password \r\n");

      //Erase All Files
      retcode = RY5_EraseAllFile(handle); //delete file
      if ( retcode != RY5_SUCCESS)
      {
      printf("\r\n RY5_EraseAllFile()error , error code : %08X \r\n", retcode);
      return;
      }

      printf("\r\n RY5_EraseAllFile() success , delete file successfully");

      //close
    retcode = RY5_Close(handle, TRUE); //close the dongle and clear the safe flag
      if ( retcode != RY5_SUCCESS)
      {
        printf("\r\n RY5_Close()error , error code : %08X \r\n", retcode);
        return;
      }

      printf("\r\n RY5_Close()  , the No. %d dongle was closed \r\n", i);
    }
```

7.1.20 RY5_GenRsaKey

DWORD WINAPI RY5_GenRsaKey(RY_HANDLE handle, int kid, BYTE* pPubBakup, BYTE* pPriBakup);

Objective:

Generate RSA key pair**.** RY5_Open must be called to open the device and the developer password needed to be verified before this function is called.

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
kid	[in]The key pair ID, ranging from 0 to 8
*pPubBakup	[out]The public key buffer pointer, output 140 bytes public key as backup

*pPriBakup [out]The private key buffer pointer, output 140 bytes private key as backup

Note:

- ■key pair ID. It ranges from 0 to 8. 0 means upgrading key pair and 1-8 means key pair ID.
- ■If *pPubBakup or *pPriBakup is NULL, it means no backup
- ■RSA key pair generated by the function only supports 1024 bit

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure, For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handle
DWORD retcode;            //error code
int count = 0;            //the found dongle count
char vendorID[10]="A8C3E6FD";   //vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() ，the found dongle count：%d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open the dongle whose No. is ‘i’
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() success， open the No. %d dongle\r\n", i);

    //Verify DevPin
    char pInPin[30]="123456781234567812345678"; //developer password
    int pRemainCount=0;  //pin remain count
   //verify developer password
    retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_VerifyDevPin()error , error code : %08X ,
         pRemainCount=%d\r\n ", retcode,pRemainCount);
        return;
    }

    printf("\r\n RY5_VerifyDevPin() , verify developer password \r\n");

    //Rsa Key
    int kid = 1;  // key pair ID
    retcode = RY5_GenRsaKey(handle,kid,NULL,NULL); // generate key pair 1
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_GenRsaKey()error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_GenRsaKey() success\r\n");

    //close
retcode = RY5_Close(handle, TRUE);//close the dongle and clear the safe flag
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Close()error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Close() , the No. %d dongle was closed. \r\n", i);
}
```

7.1.21 RY5_SetRsaKey

DWORD WINAPI RY5_SetRsaKey(RY_HANDLE handle, int kid, BYTE* pPubKey, BYTE* pPriKey);

Objective:

Set RSA key pairs. RY5_Open must be called to open the device and the developer password needs to be verified before this function is called.

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
kid	[in]Key pair ID ranges from 0 to 8.
*pPubKey	[in]Public key buffer pointer. 140 bytes public key is input for writing.
*pPriKey	[in]Private key buffer pointer. 340 bytes private key is input for writing

Note:

- The key pair ID. It ranges from 0 to 8. 0 means upgrading key pair and 1-8 means key pair ID
- If *pPubKey or *pPriKey is NULL, it means no writing.

- RSA only supports 1024 when this function is called.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handle
DWORD retcode;             //error code
int count = 0;             //the found dongle count
char vendorID[10]="A8C3E6FD";   //vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error , error code : %08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find()  , the found dongle count : %d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  // open the dongle whose No. is ‘i’
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error , error code : %08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open() success ,  open the No. %d dongle\r\n", i);

    //Verify Developer password
    char pInPin[30]="123456781234567812345678"; //developer password
    int pRemainCount=0;  //developer password remain count
    //verify developer password
    retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_VerifyDevPin()error , error code : %08X,
         pRemainCount=%d\r\n ", retcode,pRemainCount);
        return;
    }

    printf("\r\n RY5_VerifyDevPin() success , verify developer password
    successfully.\r\n");
```

```
                //set Rsa Key
                int kid = 1;  // key pair ID
                BYTE
pubBuf[140]={0x00,0x00,0x00,0x80,0xa7,0x0e,0xb4,0x0a,0x41,0x2f,0x09,0x1f,0x79,0xc5,
0xe1,0x5e,
        0xd4,0xe6,0xaa,0xca,0x71,0xae,0xa0,0x8c,0x8d,0x99,0x7d,0x44,0x17,0x88,0xd6,0
x1c,
        0xf8,0x76,0x0c,0x38,0x70,0xcc,0x59,0xa1,0xed,0xd1,0x47,0x49,0x1a,0xcd,0x0c,0
xf8,
        0x92,0xbb,0x59,0x65,0x5e,0x5c,0x1a,0x68,0x25,0xe8,0x47,0xe2,0x1f,0x4c,0xe8,0
x2e,
        0x66,0x39,0x90,0xf7,0xb0,0x8d,0x63,0xcd,0xcc,0xa0,0x93,0xe9,0xab,0xae,0xf5,0
x99,
        0x7b,0xa8,0x2f,0x86,0x71,0xe0,0x7c,0x81,0x7f,0x47,0xc9,0xe0,0x0e,0xe2,0x7a,0
x99,
        0x10,0x9f,0x7a,0x7f,0xac,0x96,0x0d,0x5d,0xcf,0xf7,0xb4,0x6b,0xbf,0xa4,0xa4,0
x87,
        0x84,0x81,0xb9,0x2c,0x66,0x32,0x7a,0xdc,0xcc,0x3f,0x4b,0x37,0x25,0x10,0xe5,0
x75,
        0xe5,0xcb,0x32,0x93,0x00,0x00,0x00,0x04,0x00,0x01,0x00,0x01,
        };  //public key

        BYTE
priBuf[340]={0x01,0x00,0x00,0x40,0xd3,0x6b,0xd4,0x80,0xac,0xe7,0x95,0x15,0xdc,0xc2,
0x2d,0xa0,
        0xae,0xe3,0xd9,0x79,0xd0,0x63,0x9d,0xad,0xab,0x44,0xe5,0x56,0xca,0xb2,0x07,0
x18,
        0xbd,0x59,0x2b,0xeb,0x34,0x4f,0xc6,0xe6,0x1f,0x99,0xd0,0x0b,0x71,0xab,0x03,0
xe3,
        0x87,0x62,0xe7,0xb0,0xbd,0x66,0xbb,0xe8,0x06,0xbf,0x9d,0x40,0xe3,0x62,0x32,0
xdd,
        0xd6,0x53,0x3b,0x29,0x00,0x00,0x00,0x40,0xca,0x48,0x30,0x79,0xfa,0x58,0x27,0
x62,
        0x18,0x8f,0x50,0x02,0x55,0x74,0x61,0x30,0xa1,0x96,0x31,0x67,0xa7,0x16,0x38,0
x40,
        0x75,0x67,0xcf,0xdd,0xc3,0xe6,0x4f,0x1e,0x05,0xd7,0xe3,0xaa,0x90,0xc7,0x14,0
x4e,
        0x23,0x02,0x72,0xbe,0x88,0x3c,0xbf,0xd1,0xe5,0x5d,0xcd,0x0f,0xb7,0xc6,0xf8,0
x08,
        0x18,0xd5,0x21,0x75,0x6f,0x8c,0x33,0x5b,0x00,0x00,0x00,0x40,0x10,0x57,0xc3,0
x83,
        0x76,0xda,0x27,0xf2,0x1a,0xb0,0x7f,0xea,0x6d,0xda,0x7b,0x13,0x29,0x82,0xa9,0
xcb,
        0xa4,0xf3,0x9c,0x6e,0xcb,0x18,0x5b,0xe8,0x13,0xaa,0xf2,0xf4,0x29,0x85,0x09,0
x30,
        0xfc,0xbc,0x5d,0x12,0xa5,0x64,0xbc,0xd6,0x37,0xbe,0xf3,0x7d,0xfc,0x06,0x16,0
xfc,
        0x66,0x99,0x25,0x44,0xc0,0xce,0xf4,0x73,0xa5,0x7f,0xa5,0x91,0x00,0x00,0x00,0
x40,
        0x30,0x7e,0x13,0xd8,0xfa,0x11,0xee,0x07,0x0f,0x3a,0x34,0x05,0x0b,0x35,0x66,0
x23,
        0x41,0x69,0x7f,0x18,0x89,0x2e,0xc3,0x77,0x78,0x6d,0xd5,0xa0,0x25,0x59,0x6d,0
xf8,
    0x15,0x90,0x86,0xcc,0x99,0xfe,0x8f,0xb8,0x97,0x9d,0xfb,0xa1,0xff,0x89,0x08,0
xb7,
    0x29,0xd4,0x4d,0xd7,0x89,0xc8,0x12,0x27,0xe6,0xc4,0x19,0x4c,0x4c,0xdc,0x80,0
x67,
    0x00,0x00,0x00,0x40,0x9e,0x20,0x54,0x80,0x49,0x6e,0xc7,0x8d,0x39,0x87,0x61,0
x5a,
    0xc5,0x7f,0xb4,0x47,0xb2,0xaa,0x21,0x7b,0xce,0x73,0x4f,0x3d,0xa5,0x35,0x39,0
xb9,
    0x69,0xdc,0x1d,0x3b,0xe4,0x90,0xa9,0xad,0xb0,0x48,0xf5,0x46,0x8e,0x80,0x83,0
xa2,
    0x3d,0x3f,0xc3,0x18,0xaf,0xdb,0x72,0xf6,0x3b,0x3a,0xe7,0x60,0xe0,0x2a,0x54,0
x2e,
    0xff,0xb0,0x1a,0x91,
    };  //private key
      retcode = RY5_SetRsaKey(handle,kid,pubBuf,priBuf); //generate key pair 1
        if ( retcode != RY5_SUCCESS)
        {
          printf("\r\n RY5_SetRsaKey()error , error code : %08X \r\n", retcode);
          return;
        }

        printf("\r\n RY5_SetRsaKey() success\r\n");

      //close
      retcode = RY5_Close(handle, TRUE); //close the dongle and clear the safe flag
        if ( retcode != RY5_SUCCESS)
        {
          printf("\r\n RY5_Close()error , error code : %08X  \r\n", retcode);
          return;
        }

        printf("\r\n RY5_Close() success , the No. %d dongle was closed.\r\n", i);
      }
```

7.1.22 RY5_PublicEncrypt

DWORD WINAPI RY5_PublicEncrypt(RY_HANDLE handle, int kid, BYTE* pBuf, int len);

Objective:

RSA encryption by using public key. RY5_Open must be called to open the device before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
kid	[in]The key pair ID that ranges from 1 to 8.
*pBuf	[in]The pointer point to the buffer for storing the data to be encrypted
len	[in]The data length must be a multiple of 128.

Note:

- The key pair ID ranging from 1 to 8.
- The data length must be a multiple of 128.
- For RSA, RY5_PublicEncrypt is usually for encrypting, but also working for decrypting of public key.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

Check RY5_PrivateDecrypt example.

7.1.23 RY5_PrivateDecrypt

DWORD WINAPI RY5_PrivateDecrypt(RY_HANDLE handle, int kid, BYTE* pBuf, int len);

Objective:

RSA decryption by using private key. RY5_Open must be called to open the device before this function is called**.**

Parameters:

handle [in]Device handle, as same as the handle returned by RY5_Open

kid [in]The key pair ID that ranges from 1 to 8.

*pBuf [out]The pointer points to the buffer for storing the data to be decrypted

len [in]The data length must be a multiple of 128.

Note:

- The key pair ID ranging from 1 to 8.
- The data length must be a multiple of 128.
- For RSA, RY5_PublicDecrypt is usually for decryption, but also working for decrypting of private key.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
        RY_HANDLE handle = 0;      //device handle
        DWORD retcode;            //error code
        int count = 0;            //the found dongle count
        char vendorID[10]="A8C3E6FD";   //vendor ID
        int i;
        //find
        retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID’s dongle

        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Find() ，the found dongle count：%d\r\n", count);

        for(i=1; i<=count; i++)
        {
            //open
            retcode = RY5_Open(&handle, i);  // open the dongle whose No. is ‘i’
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Open() success， open dongle\r\n", i);

            //Public Encrypt
            int kid = 1;  //key pair ID
            int len =128;  //data length
            BYTE buf[128]="hello";
            retcode = RY5_PublicEncrypt(handle,kid,buf,len); //RSA public key
                                                      //encryption
            if ( retcode != RY5_SUCCESS)
            {
            printf("\r\n RY5_PublicEncrypt()error，error code：%08X \r\n", retcode);
            return;
            }

            printf("\r\n RY5_PublicEncrypt() success\r\n");

            //Public Encrypt
            BYTE rbuf[128]; //store the decrypted data
            memset(rbuf, 0, sizeof(rbuf));
            retcode = RY5_PrivateDecrypt(handle,kid,rbuf,len); //RSA private key
                                                        // decryption
            if ( retcode != RY5_SUCCESS)
            {
            printf("\r\n RY5_PrivateDecrypt()error，error code：%08X \r\n", retcode);
            return;
            }

            printf("\r\n RY5_PrivateDecrypt() success， the decrypted
                   data：%08X\r\n",rbuf);

            //close
     retcode = RY5_Close(handle, TRUE);  //close the dongle and clear the safe flag
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Close()error，error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Close() success， the No. %d dongle was closed\r\n", i);
         }
```

7.1.24 RY5_MD5

DWORD WINAPI RY5_MD5(RY_HANDLE handle, BYTE* pBuf, int len, BYTE* pMD5);

Objective:

Use MD5 for encryption. RY5_Open must be called to open the device before this function is called**.**

Parameters:

handle [in]Device handle, as same as the handle returned by RY5_Open

*pBuf [in]The pointer points to the buffer for storing data

len [in]The data length

*pMD5 [out]The pointer points to the buffer for storing the calculation result whose length is 16 bytes

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
            RY_HANDLE handle = 0;      //device handle
            DWORD retcode;            //error code
            int count = 0;            //the found dongle count
            char vendorID[10]="A8C3E6FD";   //vendor ID
            int i;

            //find
            retcode = RY5_Find(vendorID, &count); //find the certain vendor ID’s dongle

            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Find()  ，the found dongle count：%d\r\n", count);

            for(i=1; i<=count; i++)
            {
```

```
        //open
        retcode = RY5_Open(&handle, i);  // open dongle i
        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Open() success， open the No. %d dongle\r\n", i);

        //MD5
        int len =5;  //the data length
        BYTE buf[10]="hello"; //the data to be encrypted
        BYTE md5Buf[20];
        memset(md5Buf, 0, sizeof(md5Buf));
        retcode = RY5_MD5(handle,buf,len,md5Buf); //MD5
        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_MD5()error，error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_MD5() success, the encrypted data：%08X\r\n",md5Buf);

        //close
    retcode = RY5_Close(handle, TRUE);//close the dongle and clear the safe flag
        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Close()error，error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Close() success， the No. %d dongle was closed\r\n", i);
    }
```

7.1.25 RY5_SHA1

DWORD WINAPI RY5_SHA1(RY_HANDLE handle, BYTE* pBuf, int len, BYTE* pSHA1);

Objective:

Use SHA1 for encryption. RY5_Open must be called to open the device before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
*pBuf	[in]The pointer points to the buffer for storing data
len	[in]The data length
*pSHA1	[out]The pointer points to the buffer for storing the calculation result whose length is 20 bytes.

Note:

- Hash result will be saved in COS and a buffer in COS is used to save the last hashing result.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handle
DWORD retcode;            //error code
int count = 0;            //the found dongle count
char vendorID[10]="A8C3E6FD";   //vendor ID
int i;

//find
retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID’s dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
    return;
}

printf("\r\n RY5_Find() ，the found dongle count：%d\r\n", count);

for(i=1; i<=count; i++)
{
    //open
    retcode = RY5_Open(&handle, i);  //
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_Open(), the No. %d dongle was closed \r\n", i);

    //SHA1
    int len =5;
    BYTE buf[10]="hello";
    BYTE sha1Buf[30];
    memset(sha1Buf, 0, sizeof(sha1Buf));
    retcode = RY5_SHA1(handle,buf,len,sha1Buf); //SHA1
    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_SHA1()error，error code：%08X \r\n", retcode);
        return;
    }

    printf("\r\n RY5_SHA1() ， encrypted data：%08X\r\n",sha1Buf);
```

```
        //close
retcode = RY5_Close(handle, TRUE);  //close the dongle and clear the safe flag
        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Close()error , error code : %08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Close(), the No. %d dongle was closed. \r\n", i);
    }
```

7.1.26 RY5_Set3DESKey

DWORD WINAPI RY5_Set3DESKey(RY_HANDLE handle, int kid, BYTE* pKey);

Objective:

RY5_Open must be called to open the device and the developer password needs to be verified before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
kid	[in]Key pair ID that ranges from 1 to 8.
*pKey	[in]The pointer points to the buffer for storing key value.

Note:

- The key pair ID ranging from 1 to 8
- The key length is fixed in 16 bytes

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handle
    DWORD retcode;             //error code
    int count = 0;             //the found dongle count
    char vendorID[10]="A8C3E6FD";   //vendor ID
    int i;

    //find
    retcode = RY5_Find(vendorID, &count);    //find the certain vendor ID’s dongle

    if ( retcode != RY5_SUCCESS)
    {
        printf("\r\n RY5_Find()error , error code : %08X \r\n", retcode);
```

```
            return;
        }

        printf("\r\n RY5_Find() ，the found dongle count：%d\r\n", count);

        for(i=1; i<=count; i++)
        {
            //open
            retcode = RY5_Open(&handle, i);  // open the dongle i
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Open() success， open the NO. %d dongle\r\n", i);

            //Verify Developer password
            char pInPin[30]="123456781234567812345678"; //developer password
            int pRemainCount=0;  //password remain count
            //verify developer password
            retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_VerifyDevPin()error，error code：%08X
                 pRemainCount=%d\r\n ", retcode, pRemainCount);
                return;
            }
            printf("\r\n RY5_VerifyDevPin() success，verify developer password
              success.\r\n");

            //set 3DES Key
            int kid = 1;  // key pair ID
            BYTE buf[16];// key
            for(int j=0; j<sizeof(buf); j++)
            {
                buf[j] = j;
            }
            retcode = RY5_Set3DESKey(handle,kid,buf); //3DES key pair 1
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Set3DESKey()error ,error code :%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Set3DESKey() \r\n");

            //close
        retcode = RY5_Close(handle, TRUE);//close the dongle and clear the safe flag
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Close()error，error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Close() success , close the NO. %d dongle\r\n", i);
        }
```

7.1.27 RY5_3DES

DWORD WINAPI RY5_3DES(RY_HANDLE handle, int kid, int flag, BYTE* pInBuf, int len);

Objective:

Use 3DES to encrypt /decrypt data. RY5_Open must be called to open the device before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
kid	[in]Key pair ID that ranges from 1 to 8.
flag	[in]Encrypt/decrypt flag.
*pInBuf	[in/out]The buffer pointer stores data, input the cipher text and output the plain text
len	[in]The data length must be a multiple of 8

Note:

■The parameter kid is set for key pair ranging from 1 to 8.

■The data length must be a mutilpe of 8. The decrypted data will be padded to get the plain text. The decrypt mode is ECB. For other mode like CBC, please modify youself.

■Flag definition:

#define FLAG_ENCODE   0   //encryption

#define FLAG_DECODE   1   //decryption

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

```
    RY_HANDLE handle = 0;      //device handle
    DWORD retcode;             // error code
    int count = 0;             //the found dongle count
    char vendorID[10]="A8C3E6FD";   //vendor ID
    int i;

    //find
    retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID’s dongle
```

```
        if ( retcode != RY5_SUCCESS)
        {
            printf("\r\n RY5_Find()error，error code：%08X \r\n", retcode);
            return;
        }

        printf("\r\n RY5_Find() ，the found dongle count：%d\r\n", count);

        for(i=1; i<=count; i++)
        {
            //open
            retcode = RY5_Open(&handle, i);  // open dongle i
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Open() success， open the No. %d dongle\r\n", i);

            //3DES
            int kid = 1;  //key pair ID
            BYTE buf[16];
            int len=16;
            for(int j=0; j<sizeof(buf); j++)
            {
                buf[j] = j;
            }

            retcode = RY5_3DES(handle,kid,FLAG_ENCODE,buf,len); // 3DS encode
            if ( retcode != RY5_SUCCESS)
            {
           printf("\r\n RY5_3DES(FLAG_ENCODE)error，error code：%08X \r\n", retcode);
            return;
            }
            printf("\r\n RY5_3DES(FLAG_ENCODE) success\r\n");

            memset(buf, 0, sizeof(buf));
            retcode = RY5_3DES(handle,kid,FLAG_DECODE,buf,len); //3DES decode
            if ( retcode != RY5_SUCCESS)
            {
        printf("\r\n RY5_3DES(FLAG_DECODE)error，error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_3DES(FLAG_DECODE) success\r\n");

            //close
    retcode = RY5_Close(handle, TRUE);  //close the dongle and clear the safe flag
            if ( retcode != RY5_SUCCESS)
            {
                printf("\r\n RY5_Close()error，error code：%08X \r\n", retcode);
                return;
            }

            printf("\r\n RY5_Close() success，  the No．%d dongle was closed\r\n", i);
        }
```

7.1.28 RY5_Update

DWORD WINAPI RY5_Update(HANDLE handle, BYTE* pbuf, int len);

Objective:

Remote update the data on dongle. RY5_Open must be called to open the device before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
*pBuf	[in/out]The pointer points to the buffer for storing the data to be updated
len	[in]The data length

Note:

■The data structure of update package:

```
typedef struct
{
    BYTE    sn[8];          //hardware ID, FFFFFFFFFFFFFFFF represents no restriction of hardware ID
    BYTE    type;           //0: public key file    1: private key file    2: key file
                            //3：   the 1024 byte read only area
    BYTE    item;           //Item: 0-8 (the key file for updating )
    WORD    offset;         //offset address
    BYTE    len;            // the read-in data length (<=115 byte)
    BYTE    data[115];      // the 115 byte data block
} UpdateData;
```

■The private key will be called to decrypt the update package when updating ROCKEY5 TIME. It needs to be ensured that the private key for remote updating is located in the dongle. The user can use the user tool in ROCKEY5 TIME to generate and import or use exsited API. Note that the remote public key is not recommended to write to the dongle.

■For more detail about the remote update, please check 3.2.7, here introduces the APIs for remote updating.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

Example:

Ckeck RY5_GenUpdatePacket example

7.1.29 RY5_GenUpdatePacket

DWORD WINAPI RY5_GenUpdatePacket(RY_HANDLE handle,

char* pLicSN,

int type,

int kid_offset,

BYTE* pbuf,

int len,

BYTE* pUPubKey,

BYTE* pOutData,

int* pOutLen);

Objective:

Generate the remote updating package. RY5_Open must be called to open the device and the verification is needed for developer pin before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
*pLicSN	[in]The pointer points to the buffer for storing the hardware serial number
type	[in]The data type for updating
kid_offset	[in]The group ID of key pair to be updated or offset address。
*pbuf	[in] The pointer points to the buffer for storing the updating data.
len	[in]The data length for updating
*pUPubKey	[in]The buffer pointer stores the public key data
*pOutData	[out]The pointer points to the buffer for storing the returned updating package

*pOutLen [out]The pointer points to the buffer for storing the returned updating package length

Note:

- If the package is only for certain dongle, the parameter *pLicSN needs to be set as its dongle's hardware serial number. Else if *pLicSN is NULL, there is no restriction for hardware serial number.
- Definition of parameter type and the data type to update:

  #define Data_TYPE_PubKey 0 //public key data

  #define Data_TYPE_PriKey 1 //private key data

  #define Data_TYPE_3DesKey / Data_TYPE_AESKey 2 // 3DES and AES share the same key data

  #define Data_TYPE_Memory 3 //8192 bytes data area

  #define Data_TYPE_Time 4 //update time limits. *pbuf value is the time (4 bytes) + time type (1 byte) + update time stamp (4 bytes)
- The parameter kid_offset have different presentations due to the type:

  If type = Data_TYPE_Memory, kid_offset represents offset adderss of user space (8192 bytes)

  If type is 0\1\2, kid_offset represents the key pair ID to be update ranging 1 to 8.
- If the data type to be update is public key data (type= Data_TYPE_PubKey), the parameter len must be 140 bytes. Else if the data type to be updated is private key data (type= Data_TYPE_PriKey), len must be 340 bytes; And if the data type is 3DES key data (type= Data_TYPE_3DesKey), len must be 16 bytes.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure, for more detail, check the error list in section 7.2.

Example:

```
RY_HANDLE handle = 0;      //device handle
DWORD retcode;             //error code
int count = 0;             //the found dongle count
char vendorID[10]="A8C3E6FD";   //vendor ID
int i;
char pInPin[30]="1234567812345678l2345678"; //developer password
int pRemainCount=0;  //developer password retry times

//find
retcode = RY5_Find(vendorID, &count);   //find the certain vendor ID's dongle

if ( retcode != RY5_SUCCESS)
{
    printf("\r\n RY5_Find()error , error code : %08X \r\n", retcode);
    return;
}
```

```
            printf("\r\n RY5_Find() ，the found dongle count：%d\r\n", count);

            for(i=1; i<=count; i++)
            {
                //open
                retcode = RY5_Open(&handle, i);  // open dongle
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_Open()error，error code：%08X \r\n", retcode);
                    return;
                }

                printf("\r\n RY5_Open() success， open the No. %d dongle\r\n", i);
               //verify developer pin
                retcode = RY5_VerifyDevPin(handle, pInPin, &pRemainCount);
               if ( retcode != RY5_SUCCESS)
                {
                  printf("\r\n  RY5_VerifyDevPin() error，error code：%08X
                   pRemainCount=%d\r\n ", retcode,pRemainCount);
                  return;
                }

                //the updating data
                BYTE
buf[140]={0x00,0x00,0x00,0x80,0xa7,0x0e,0xb4,0x0a,0x41,0x2f,0x09,0x1f,0x79,0xc5,0xe
1,0x5e,

    0xd4,0xe6,0xaa,0xca,0x71,0xae,0xa0,0x8c,0x8d,0x99,0x7d,0x44,0x17,0x88,0xd6,0x1c
,

    0xf8,0x76,0x0c,0x38,0x70,0xcc,0x59,0xa1,0xed,0xd1,0x47,0x49,0x1a,0xcd,0x0c,0xf8
,

    0x92,0xbb,0x59,0x65,0x5e,0x5c,0x1a,0x68,0x25,0xe8,0x47,0xe2,0x1f,0x4c,0xe8,0x2e
,

    0x66,0x39,0x90,0xf7,0xb0,0x8d,0x63,0xcd,0xcc,0xa0,0x93,0xe9,0xab,0xae,0xf5,0x99
,

    0x7b,0xa8,0x2f,0x86,0x71,0xe0,0x7c,0x81,0x7f,0x47,0xc9,0xe0,0x0e,0xe2,0x7a,0x99
,

    0x10,0x9f,0x7a,0x7f,0xac,0x96,0x0d,0x5d,0xcf,0xf7,0xb4,0x6b,0xbf,0xa4,0xa4,0x87
,

    0x84,0x81,0xb9,0x2c,0x66,0x32,0x7a,0xdc,0xcc,0x3f,0x4b,0x37,0x25,0x10,0xe5,0x75
,
                    0xe5,0xcb,0x32,0x93,0x00,0x00,0x00,0x04,0x00,0x01,0x00,0x01,
                };

                int len;

                // generate RSA key pair for remote update
        // the keys will store in pubBuf and priBuf. For future generating updating package
        // and keys , they need to be saved.
        // the public key need to be matched with the private key in dongle when updating
                BYTE pubBuf[140] = {0};
                BYTE priBuf[340] = {0};

                retcode = RY5_GenRsaKey(handle,0,pubBuf,priBuf);
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_GenRsaKey()error，error code：%08X \r\n", retcode);
                    return;
                }

                BYTE out_buf[1024];
                memset(out_buf, 0, sizeof(out_buf));



    retcode=RY5_GenUpdatePacket(handle,NULL,Data_TYPE_PubKey,1,buf,140,pubBuf,
        out_buf,&len); //Generate update package
                if ( retcode != RY5_SUCCESS)
                {
            printf("\r\n RY5_GenUpdatePacket()error，error code：%08X \r\n", retcode);
            return;
                }
                printf("\r\n RY5_GenUpdatePacket() success\r\n");

                //Update
                //using private key for updating
                retcode = RY5_Update(handle, out_buf, len); //updating
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_Update()error，error code：%08X \r\n", retcode);
                    return;
                }
                printf("\r\n RY5_Update() success\r\n");

                //close
        retcode = RY5_Close(handle, TRUE);  //close the dongle and clear the safe flag
                if ( retcode != RY5_SUCCESS)
                {
                    printf("\r\n RY5_Close()error，error code：%08X \r\n", retcode);
                    return;
                }

                printf("\r\n RY5_Close() success， close dongle\r\n", i);
            }
```

7.1.30 RY5_AES

DWORD WINAPI RY5_AES(RY_HANDLE handle, int kid, int flag, BYTE* pBuf, int len);

Objective:

Use AES for encrypting/decrypting data. RY5_Open must be called to open the device before this function is called**.**

Input parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open.
kid	[in]The key pair ID that ranges from 1 to 8.
flag	[in] 0 represents encryption and 1 represents decryption.
*pBuf	[in/out]The pointer points to the buffer for storing the encrypted/decrypted data.
len	[in]The data length that must be a multiple of 16.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

7.1.31 RY5_SetAESKey

DWORD WINAPI RY5_SetAESKey(RY_HANDLE handle, int kid, BYTE* pKey);

Objective:

Set AES keys. RY5_Open must be called to open the device before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
kid	[in]The key pair ID that ranges from 1 to 8.
*pKey	[in/out]The pointer points to the buffer for storing 16 byte AES keys

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

7.1.32 RY5_GetRTCTime

DWORD WINAPI RY5_GetRTCTime(RY_HANDLE handle,struct tm* pRTC);

Objective:

Get the dongle time. RY5_Open must be called to open the device before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
*pRTC	[in/out]It points to the struct tm defined in Windows time.h

Note:

```
struct tm {
        int tm_sec;     /* seconds after the minute - [0,59] */
        int tm_min;     /* minutes after the hour - [0,59] */
        int tm_hour;    /* hours since midnight - [0,23] */
        int tm_mday;    /* day of the month - [1,31] */
        int tm_mon;     /* months since January - [0,11] */
        int tm_year;    /* years since 1900 */
        int tm_wday;    /* days since Sunday - [0,6] */
        int tm_yday;    /* days since January 1 - [0,365] */
        int tm_isdst;   /* daylight savings time flag */
        };
```

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

7.1.33 RY5_SetTimeLimit

DWORD WINAPI RY5_SetTimeLimit(RY_HANDLE handle, int value, BYTE policy);

Objective:

Set the time limits. RY5_Open must be called to open the device and the developer password needs to be verified before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open。
value	[in]The remaining hours of UTC time. The length is 32 bits. If it is 0, it means no time limits.
policy	[in]The flag of limitation of time; 0 means the limit UTC time; 1 means the limit hours.

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

7.1.34 RY5_GetTokenType

DWORD WINAPI RY5_GetTokenType(RY_HANDLE handle, int* type);

Objective:

Get the token type. RY5_Open must be called to open the device and the developer password needs to be verified before this function is called**.**

Parameters:

handle	[in]Device handle, as same as the handle returned by RY5_Open
*type	[in/out]The token type; 1 represents ROCKEY5 TIME and 2 represents others

Return value：

Success returns RY5_SUCCESS（0x00000000. Other return means failure. For more detail, check the error list in section 7.2.

7.2 Error Code

```
#define    RY5_SUCCESS                    0x00000000    //successfully done
#define    RY5_NOT_FOUND                  0xF0000001   // device Not found
#define    RY5_INVALID_PARAMETER          0xF0000002    // invalid parameter
#define    RY5_COMM_ERROR                 0xF0000003    // comunication error
#define    RY5_INSUFFICIENT_BUFFER        0xF0000004    //not enough buffer space
#define    RY5_NO_LIST                    0xF0000005    //the device list not find
#define    RY5_DEVPIN_NOT_CHECK           0xF0000006    // no verification of developer pin
#define    RY5_USERPIN_NOT_CHECK          0xF0000007    //no verification of user pin
#define    RY5_RSA_FILE_FORMAT_ERROR       0xF0000008   //error format of RSA file
#define    RY5_DIR_NOT_FOUND              0xF0000009    //directory not found
#define    RY5_ACCESS_DENIED              0xF000000A    //access denied
#define    RY5_ALREADY_INITIALIZED        0xF000000B    //Initialized
#define    RY5_INCORRECT_PIN              0xF0000C00    //incorrect pin
#define    RY5_DF_SIZE                    0xF000000D    //not enough space for target directory
#define    RY5_FILE_EXIST                 0xF000000E    // the file already exists
#define    RY5_UNSUPPORTED                0xF000000F    //unsupported function or system
#define    RY5_FILE_NOT_FOUND             0xF0000010    //file not found
#define    RY5_ALREADY_OPENED             0xF0000011    //already opened
```

```
#define     RY5_DIRECTORY_EXIST            0xF0000012     //the directory already exists
#define     RY5_CODE_RANGE                 0xF0000013     //memory overflow
#define     RY5_INVALID_POINTER            0xF0000014     // the invalid pointer of virtual machine
#define     RY5_GENERAL_FILESYSTEM         0xF0000015     //the error of general file system
#define     RY5_OFFSET_BEYOND              0xF0000016     //the offset beyond the size of file
#define     RY5_FILE_TYPE_MISMATCH         0xF0000017     //mismatch of the file type
#define     RY5_PIN_BLOCKED                0xF0000018     // pin locked
#define     RY5_INVALID_HANDLE             0xF0000019     // invalid handle
#define     RY5_ERROR_UNKNOWN              0xFFFFFFFF     //    unknown error
#define     RY5_FUNC_EXPIRED               0x00000017     // COS expired
#define     RY5_HW_CLOCK_BROKEN            0x00000018     //hardware clock broken
#define     0xf0000cXX                     0xf0000cXX     //the failure of the pin verification.xx
//represents the remaining time ranging from 0 to 0xFF. 0xFF means no limits.
```