W1680 IC卡读写器

用户开发手册

(V1.2)

北京握奇数据系统有限公司

二〇一二年五月

目录

CCC 认证声明

此产品为 A 级产品，在生活环境中，该产品可能会造成无线电干扰。在这种情况下，可能需要用户对其干扰采取切实可行的措施。

一、功能和性能介绍

W1680 读写器即原 CRW-VIps 读卡器，是对多卡座版的 CRW-V plus 读卡器的简化版本，主卡座支持符合 ISO/IEC 7816 协议的智能卡和多种常用存储卡，具有结构小巧、方便携带、功耗低、性价比高等优点。该读卡器采用标准 USB 接口通讯，同时可支持 PC/SC 协议。

1. 主要功能

- 主卡座支持符合 ISO/IEC 7816 协议的 CPU 卡
- CPU 卡通讯速率 9600/19200/38400/56K/112K
- 主卡座支持多种常用存储卡，包括 AT24C01/02/04/08/16 及 AT24C32/64、SLE4432/42、SLE4418/28、AT88SC102、AT88SC1604、AT88SC1608、AT88SC153
- 主卡座可以以 5V/3V/1.8V *（注 1）*三种电压操作卡片
- 支持 PC/SC 协议，通过该接口可实现对 CPU 卡和存储卡的操作
- 支持标准 USB 通讯接口，符合 USB1.1 标准，通讯速率为 1.5Mbps
- SAM 卡支持符合 ISO/IEC 7816 的 CPU 卡
- 两个指示灯，对应电源状态和读写器工作状态

注 1：1.8V 的电压支持需要定制

2. 主要技术指标

参数	指标
卡片读写速率	9600/19200/38400/56K/112K
USB 通讯速率	1.5Mbps
取电方式	USB
主卡头读写次数	20 万次
工作电压	DC5V±0.3V
工作电流	有卡 <50mA 无卡 <30mA
外型尺寸	55*65*13（mm）
工作温度	-20℃ ~ 55℃
工作湿度	20% ~ 90%
平均无故障时间	5000 小时
操作系统	适用于 Windows 2000/XP/2003/VISTA/WIN7(*注 2*)

注 2：Windows 2000/XP/2003 与 VISTA/WIN7 是不同的驱动程序

3. 配套软件

- CRW-V plus USB 驱动(适用 Windows 2000/XP/2003，已经过微软徽标认证)
- CRW-V plus USB 驱动(适用于 WIN7 VISTA)
- CRW-V plus 动态库
- 完整的开发演示程序，包括 VC/VB 的源代码

4. 符合标准

- 〈ISO/IEC 7816-1/2/3〉
- 〈USB1.1 标准〉

二、软硬件安装说明

1．读写器的连接

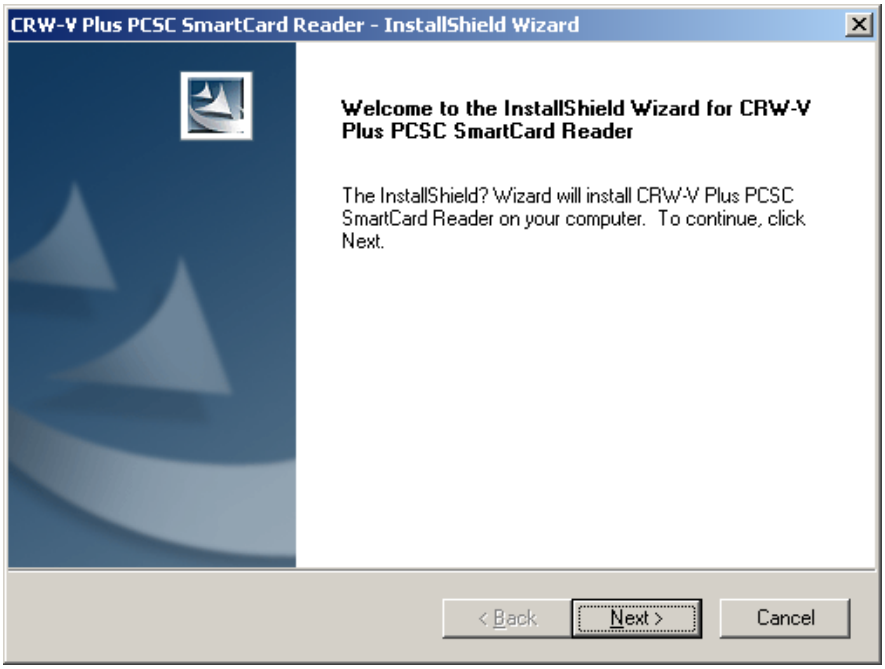
W1680 USB 读写器：

将读写器与主机通过 USB 端口相连。

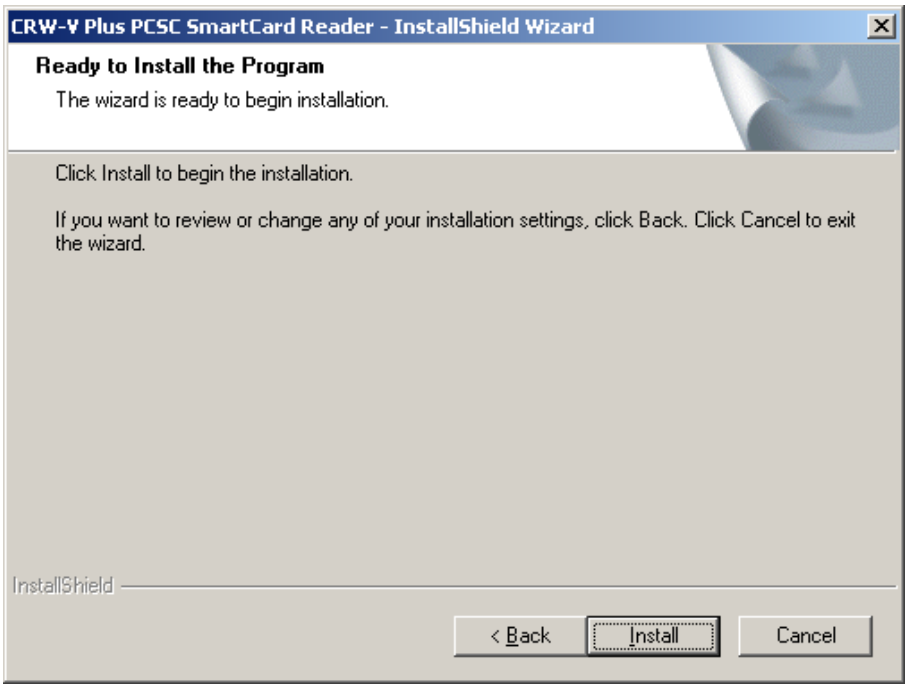
2．USB 驱动安装

本驱动与动态库相配合，以 USB 方式操作读写器；也可以不使用动态库，直接使用 PC/SC 方式通过读写器操作卡片。驱动可从 www.watchdata.com 上获得。

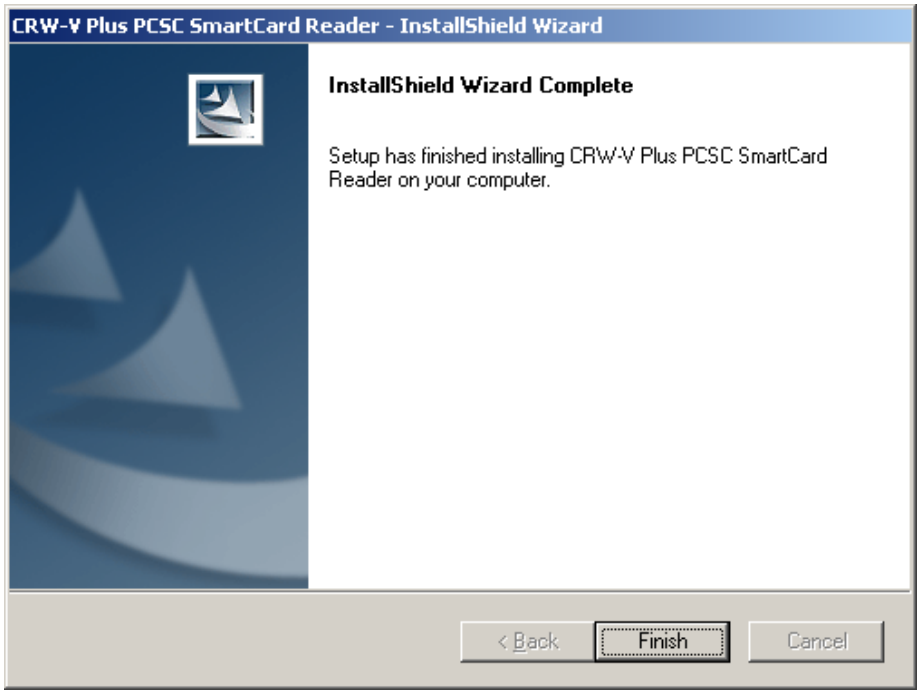
1）在“driver”文件夹中，点击“Setup.exe”安装软件，第一次安装时出现如下界面，点击“Next”按钮。



2）此时出现如下的确认安装界面，点击“Install”按钮后，程序开始安装 CRW-V plus 读写器驱动。



3）驱动安装结束后，出现如下界面，驱动安装成功，点击“Finish”退出安装界面。



说明：

若以 USB 方式操作读写器，请使用本读写器配套的动态库；

为了能够正常使用 PC/SC 方式操作读写器，请预先启动“智能卡服务”。启动的方法为：“我的电脑”右键->管理->服务和应用程序->服务->双击“Smart Card”服务，在弹出的属性框中，启动类型选择“自动”，服务状态选择“启动”，点击“确定”后，即可正常使用 PC/SC 服务。

三、CRW-Vp 读写器的通讯协议

特别说明：本手册里与命令相关的的数字默认是十六进制数据。

1.发送命令格式：

标识	字节长度	含义
NAD	1	NAD　　卡座选择 00　　主卡座的 CPU 卡 12　　主卡座的 CPU / 存储卡及读写器 13　　SAM 卡座
PCB	1	
LEN	1	数据长度，包括 CLA INS P1 P2 Lc DATA
CLA	1	指令类型
INS	1	指令码
P1	1	指令参数 1
P2	1	指令参数 2
Lc	1	输入数据长度或期望返回数据长度
DATA	0-FF *（注 2）*	输入数据
XOR	1	XOR 校验值(从 NAD 开始的所有数据做异或计算的结果)

例 1：CPU 卡选择主文件（3F00）命令

12 00 07 00 A4 00 00 02 3F 00 8C

例 2：CPU 卡复位命令

12 00 05 00 12 00 00 00 05

2.从读写器返回信息的格式

标识	字节长度	含义
NAD	1	发送命令 NAD 的半字节互换 例如发送 NAD=12，返回 NAD=21
PCB	1	
LEN	1	数据长度，包括 DATA SW1 SW2
DATA	0-FF *（注 2）*	返回数据
SW1	1	状态字节 1
SW2	1	状态字节 2
XOR	1	XOR 校验值(从 NAD 开始的所有数据做异或计算的结果)

例 1 返回信息如下：

21 00 02 61 XX XOR

例 2 返回信息如下

21 00 LEN 3B(3F) T0 TA(i)~TD(i) T1~Tk TCK 90 00 XOR

其中 90 00 是读写器自动补上状态字节 SW1 SW2

3.动态库和 PC/SC 的处理

为了方便用户的使用，发送指令和接收信息格式中的 NAD PCB LEN 和 XOR 这四个字节由提供的动态库或驱动进行了处理，用户只需要正确设置 NAD，非 PC/SC 模式下使用动态库或 PC/SC 模式下直接发送相应的 APDU 指令，即可获得 DATA 和 SW（SW1+SW2）返回信息。如无特殊说明，本手册说明发送指令和返回信息时，不涉及 NAD PCB LEN 和 XOR 这四个字节。

注 2：动态库操作时 DATA 长度可达 FF，与终端直接连接时，串口模式下，发送命令中 DATA 最大长度为 0xFA，返回信息中 DATA 最大长度为 0xFD

四、指示灯说明

CRW-Vp 正面有两个指示灯，红灯为电源指示灯，绿灯为读写器工作状态指示灯。插入主卡时，把卡片带芯片的一面朝上，插卡到位即可。在使用过程中，两灯的指示说明如下：

- 红灯灭，表示设备未上电
- 红灯亮，表示设备已上电
- 绿灯灭，表示用户卡槽内无卡或卡插入不到位
- 绿灯亮，表示有卡插入用户卡槽，并处于等待读写状态
- 绿灯快速闪烁，表示读写器正在进行数据传输
- 绿灯慢速闪烁，表示读写器不能自动识别用户卡槽中的卡片

五、通用指令说明

1.除了卡片通用的标准协议指令外，读写器有其它的一些指令，主要包括对卡片的复位，修改串口波特率等，列表如下：

NAD	CLA	INS	P1	P2	Lc	Data	功能说明
12	00	12	00	00	00		基本复位指令（存储卡和 CPU 卡自动识别）
	CLA	12	00	P2	00		指定参数复位 CLA：字符等待时间，单位 100ms P2：以指定通讯速率复位卡片，具体的参数如下： 14：38400bps
	00	16	RATE	00	00		修改读写器与 PC 间的串口通讯速率，使用该指令后，通讯速率直接改变 说明：此指令仅在动态库的 ICC_tsi_api()函数中实现，相应参数如下： 00：9600bps 02：38400bps
13	00	12	00	00	00		基本复位指令（CPU 卡）
存储卡指令（12）	CLA	B0	P1	P2	Le		读存储卡（存储卡的具体应用请看下一节）
	CLA	D0	P1	P2	Le	DATA	写存储卡
	CLA	20	P1	P2	Le	DATA	校验密码
	CLA	24	P1	P2	Le	DATA	修改密码
	CLA	F8	P1	P2	Le		按位擦除

2．读写器通用指令返回状态 SW1SW2 的含义

9000	操作成功
6FF0	卡通讯失败或其它未知的错误
6D00	不识别的读写器命令
6200	指定操作的卡不存在
FFFF(-1)	PC 与读写器通讯失败

六、存储卡操作

存储卡指令要求 NAD=0x12，指令格式 **CLA INS P1 P2 Lc (Data)**

存储卡根据卡类型其相应的指令有些出入，地址编码也不尽相同，如下表所示：

卡类型	CLA	P1	P2	说明
AT24C01~16	00	P1.0~3 字节地址高位	字节地址低位	
AT24C32/64	00	字节地址高位	字节地址低位	
AT88SC1608	00	P1.0~4 区地址	字节地址	P1=08，配置区。P1=08，P2=80，表示熔丝地址
AT88SC153	00	00/01	P2.0~5，字节地址 P2.6~7，页地址	P1=01，表示熔丝地址
AT88SC102	00	字节地址高位	字节地址低位	
	10	位地址高位	位地址低位	
AT88SC1604	00	字节地址高位	字节地址低位	
	10	位地址高位	位地址低位	
SLE4418/28	00	字节地址高位	字节地址低位	
	80	字节地址高位	字节地址低位	（带保护位）
SLE4432/42	00	00	字节地址	
	80	00	保护位字节地址	
	00	01	密码区地址	

下面对存储卡的各条指令进行说明：

1. **读数据指令 INS = B0**

   根据 CLA 的值不同，功能不同

   00：字节读

   10：按位读（AT88SC102/1604）

   80：根据卡片不同，功能定义不同：

   带保护位读（4418/28），读保护位(4432/42)，读熔丝(AT88SC153 / AT88SC1608 / AT88SC102)。

2. **写数据指令 INS = D0**

   根据 CLA 值不同，功能不同

   00：字节写

   10：按位写(AT88SC102/1604)

   80：根据卡片不同，功能定义不同：

   带保护位写（4418/28），写保护位(4432/42).熔丝(AT88SC153 / AT88SC1608 / AT88SC102)

3. **按位擦除指令 INS = F8**

   P1P2 表示擦除的位地址，长度无效，只擦除指定地址区域的数据，数据区的大小由卡类型和地址所在区域决定

4. **校验密码：INS = 20**

   根据卡类型，P1 P2 Lc 的定义不同

   P2=00，表示主密码（即用户密码或传输密码）

   P2=1~8，表示分区密码的区号

   P1.0 表示读或写密码，0 表示写密码，1 表示读密码

根据不同的卡类型，Len 的长度决定密码的用途，不符合要求的密码，其结果不确定

卡类型	P1	P2	Lc	密码区说明
SLE4428	00	00	02	用户密码

SLE4442	00	00	03	用户密码
AT88SC1608	00	00	03	传输密码
	00	01	03	1 区写密码
	01	01	03	1 区读密码
	00	02	03	2 区写密码
	01	02	03	2 区读密码
	00	03	03	3 区写密码
	01	03	03	3 区读密码
	00	04	03	4 区写密码
	01	04	03	4 区读密码
	00	05	03	5 区写密码
	01	05	03	5 区读密码
	00	06	03	6 区写密码
	01	06	03	6 区读密码
	00	07	03	7 区写密码
	01	07	03	7 区读密码
	00	08	03	8 区写密码
	01	08	03	8 区读密码
	00	09	08	初始化认证操作
	01	09	08	认证密钥匙操作
AT88SC153	00	00	03	传输密码
	00	01	03	1 区写密码
	01	01	03	1 区读密码
	00	02	03	2 区写密码
	01	02	03	2 区读密码
	00	09	08	初始化认证操作
	01	09	08	认证密钥匙操作
AT88SC102	00	00	02	用户密码
	00	00	06	1 区擦除密码
	00	00	04	2 区擦除密码
AT88SC1604	00	00	02	用户密码
	00	01	02	1 区写密码
	01	01	02	1 区擦除密码
	00	02	02	2 区写密码
	01	02	02	2 区擦除密码
	00	03	02	3 区写密码
	01	03	02	3 区擦除密码
	00	04	02	4 区写密码
	01	04	02	4 区擦除密码

5. 修改密码 INS = 24

指令格式：CLA INS P1 P2 Lc OldPin NewPin

修改用户（传输）密码有效，参数设定方法同校验密码，长度 Lc 是校验密码的两倍。

6. 返回状态说明

0x9000 操作成功

0x6282 offset+len 超出卡片容量错误

0x63Cx 校验密码错误，x 为剩余可校验次数

0x6581 写操作时，检测到失败

0x6983　卡片已锁死
0x6A86　长度 len 错误
0x6B00　偏移量 offset 错误
0x6D00　无效的操作，该卡不支持指令
0x6FF0　操作失败，卡片不识别
0x9001　卡片不合法
0x9002　MEM_AT1608_Aut()函数中外部认证没有启用或认证密钥不对时的返回值

七、CRW-V plus 系列读写器动态库函数

适应操作系统：

*USB 读写器：Windows 2000/XP/2003/VISTA/WIN7 系统

适用的 IC 卡：

*符合 7816 的 CPU 卡（T=0&T=1）

*I^2C 总线存储卡：ATMEL 公司的 AT24C01/02/04/08/16、AT24C32/64

*逻辑加密卡：Infineon 公司的 SLE4418/28、SLE4432/42

*ATMEL 公司的 AT88SC102、AT88SC1604/1608

1、读写器及 CPU 卡读写接口函数

此部分是针对 CPU 卡的操作函数，推荐的函数调用顺序

CT_open	打开设备获得设备句柄
ICC_set_NAD	设置卡座 NAD，默认值为 00
ICC_reset	对 IC 卡复位
…	
ICC_tsi_api	等函数对 IC 卡(Memory 卡)进行操作
...	
CT_close	关闭打开设备的连接

1）打开 IC 卡终端端口

```
HANDLE WINAPI CT_open(
    char *name,
    unsigned int param1,
    unsigned char param2
    );
```

参数：

（A）串口读写器

name：读写器与 PC 相连的端口，可取 COM1 COM2 COM3 COM4 等

param1：波特率，可取 9600 或 38400，由读写器的速率决定

param2：奇偶校验，可为"E"偶校验，"N"无校验，读写器默认为无校验

（B）USB 读写器

name：可取 USB1 USB2 USB3 USB4 等，或者 wdcrwvp1 wdcrwvp2 wdcrwvp3 等，建议采用 wdcrwvp 方式打开。当握奇公司的 KEY 与读写器同时插入时，必须采用此方式打开。

USB序号分配原则：

1.由设备插入的先后顺序决定，第一次插入的设备为USB1，第二次插入设备为USB2依次类推；

2.当新插入USB设备时驱动从1开始查询当前未使用的序号，直到找到为止

param1:打开设备的模式 1：共享模式 2：独占模式

param2：未使用，设为 0 即可

返回值：

INVALID_HANDLE_VALUE(-1)表示打开端口失败；

其他值(大于 0)为打开的端口句柄，用于卡操作函数的 fd

示例：

```
//以下的程序为以 9600 波特率，无校验方式打开与 COM1 口连接的串口读写器
HANDLE fDev;                //定义句柄，用于保存端口句柄
char devName[5];            //用于保存端口名称
strcpy(devName, "COM1");        //获得端口名称
fDev = CT_open(devName ,9600, 'N');   //以相应的格式打开端口，并得到端口句柄
```

```
if(fDev == INVALID_HANDLE_VALUE)       //判断端口是否正确打开
{
  MessageBox("Open Device Error!");
  return;
}
```

2）关闭与 IC 卡读写器相连的端口（必须用 CT_open 打开的端口）

```
int WINAPI CT_close(
    HANDLE fd
    );
```

参数：

fd ： 为函数 CT_open 所返回的句柄

返回值：

-1：失败 0：成功

示例：

```
//以下示例为关闭以 CT_open()函数打开的读写器
int ret;       //定义 int 变量用于保存关闭函数返回值
ret = CT_close( fDev );        //关闭端口，并获得结果
if(ret == -1)      //判断端口是否正确关闭
{
  MessageBox("Close Deivce Error");
}
```

3）对设备当前激活插槽中的 IC 卡进行复位

```
unsigned WINAPI ICC_reset(
    HANDLE fd,
    unsigned char *lenr,
    unsigned char *resp
    );
```

参数：

fd ： 已打开的端口描述符

lenr ： 为对 IC 卡复位所返回的复位数据的长度

resp ： 复位的数据结果

返回值：

0x9000 成功

0x6200 无卡

0x6201 协议不认识

0xffff 通迅失败

示例：

```
//以下示例，为对 CPU 卡片进行复位
unsigned int sw;        //定义变量分别用于保存返回状态值
unsigned int lenr, resp[256];   //定义变量分别用于保存返回数据长度及数据
ICC_set_NAD(fDev,0x12); //设置NAD为0x12
sw = ICC_reset(fDev, &lenr, resp);      //执行复位指令
if(sw != 0x9000)        //判断是否执行成功
{
    … //操作失败后，用户的处理
}
```

4）从外设向 CPU 卡或读写器发送命令 APDU 并接收应答 APDU

comm的结构：CLA INS P1 P2 Lc DATA [Le] 其中DATA长度为Lc字节

resp 的结构：DATA 其中 DATA 长度为 Le 字节

```
unsigned WINAPI ICC_tsi_api(
    HANDLE fd,
    unsigned char len,
    unsigned char *comm,
    unsigned char *lenr,
    unsigned char *resp
    );
```

参数：

fd　　：　已打开的端口描述符

len　　：　命令 comm 的长度

comm　：　发向卡上的命令

lenr　　：　从卡上接收到的数据长度

resp　　：　从卡上接收到的数据

返回值：

0xffff 通迅失败（发送命令或接收返回的数据失败）

其它为从卡上返回的状态 SW1 SW2

说明：

该函数适用于所有 CPU 卡操作，一次最多可读 253 个字节，写 250 个字节

示例：

```
//以下示例为发送取版本号指令并显示
unsigned char lens;        //定义发送的数据长度变量
unsigned char lenr;        //定义保存返回数据长度变量
unsigned char comm[300];   //定义发送指令数组
unsigned char resp[300];       //定义接收数据数组
unsigned int sw;           //定义变量用于保存返回状态值
char tmpbuf[300];      //定义变量用于保存转化为字符型值的返回值
CString strDisplay;        //定义变量用于显示
ICC_set_NAD(fDev,0x12);    //设置NAD为0x12
memcpy(comm,"\x00\x19\x00\x00\x00",5);     //设置十六进制的指令
lens=5;                //设置指令长度为5
sw=ICC_tsi_api(fDev,lens,comm,&lenr,resp);     //发送指令并取得返回值
if(sw!=0x9000)     //对指令执行是否成功进行判断
{
  MessageBox("Get Reader Firmware Version Error!");
}else
{
  BinToCHex((unsigned char *)tmpbuf,resp,lenr);        //将返回值转换为字符以供显示
  tmpbuf[lenr*2]=0;
  strDisplay=CString(tmpbuf);
  MessageBox("Firmware Version is "+strDisplay);
}
```

5）从外设向 T=0 的 CPU 卡发送命令 APDU 并接收应答 APDU

```
unsigned WINAPI ICC_tsi_apiT0(
    HANDLE fd,
    unsigned int len,
    unsigned char *comm,
    unsigned int *lenr,
    unsigned char *resp
    );
```

参数：

fd　　：　已打开的端口描述符

len　　：　命令 comm 的长度

comm　：　发向卡上的命令

lenr　：　从卡上接收到的数据长度

resp　：　从卡上接收到的数据

返回值：

0xffff 通迅失败(发送命令或接收返回的数据失败

其它为从卡上返回的状态 SW1 SW2

说明：

该函数只适用于在 CRW-V(VI)读写器读写 T=0 的支持 256 字节读写的 CPU 卡，

使用该函数一次可读多达 256 字节（len=0），写多达 255 个字节的数据

示例：

同上，写入数据和返回值可达 255 字节

6）设置 CPU 卡读写地址 NAD

```
void WINAPI ICC_set_NAD(
    HANDLE fd,
    unsigned char nad
    );
```

参数：

fd　：　已打开的端口描述符

nad：　读写地址

返回值：

无

说明：

该函数使用于 CRW-V(VI)系列读写器，系统缺省值为 00

00　对主卡操作

12　对读写器或主卡操作

13　选择双卡座读写器的 SAM 卡

示例：

见函数 4）中的以下语句

```
ICC_set_NAD(fDev,0x12);    //设置NAD为0x12
```

7）检查读写器的主卡座否插入 IC 卡

```
unsigned WINAPI ICC_present(
    HANDLE fd
    );
```

参数：

fd　：　已打开的端口描述符

返回值：

0x9000　已插入 IC 卡

0x6200　没有插入卡或卡没插到位

示例：

```
//此例程为检查卡片是否存在
unsigned int sw;          //定义返回状态变量
sw = ICC_present(fDev); //检查是否插入卡操作
if(sw != 0x9000)          //判断是否执行成功
{
    …  //操作失败后，用户的处理
}
```

8）写 CPU 卡的二进制文件

```
unsigned WINAPI ICC_write_file(
    HANDLE fd,
    unsigned int offset,
    unsigned int len,
    unsigned char *data
    );
```

参数：

fd　　：　已打开的端口描述符

offset　：　二进制文件的偏移量

len　　：　要写入卡上的数据长度

data　　：　要写入卡上的数据

返回值：

0xffff　通迅失败（发送命令或接收返回的数据失败）

其它为从卡上返回的状态 SW1 SW2

说明：

该函数适用于所有 CPU 卡操作，用户必须先选择要操作的二进制文件

示例：

```
//此示例为向 CPU 卡的二进制文件写入 3 个数据，请先对 CPU 卡片复位后，选择相应二进制文件
unsigned int offset = 0;          //定义写入数据地址的偏移量变量，并赋初值
unsigned int len;             //定义写入数据长度的变量
unsigned char data[3]={0, 1, 2};     //定义写入数据变量，并赋初值
len = 3;              //写入数据的长度为 3 字节
sw = ICC_write_file(fDev, offset, len, data);   //写入二进制文件 3 字节数据操作
if(sw != 0x9000)          //判断是否执行成功
{
    … //操作失败后，用户的处理
}
```

9）读 CPU 卡的二进制文件

```
unsigned WINAPI ICC_read_file(
    HANDLE fd,
    unsigned int offset,
    unsigned int len,
    unsigned char *data
    );
```

参数：

fd　　：　已打开的端口描述符

offset　：　二进制文件的偏移量

len　　：　要读卡上的数据长度

data　　：　要读卡上的数据

返回值：

0xffff　通迅失败（发送命令或接收返回的数据失败）

其它为从卡上返回的状态 SW1 SW2

说明：

该函数适用于所有 CPU 卡操作，用户必须先选择要操作的二进制文件

示例：

```
//此示例为向 CPU 卡的二进制文件读取 3 个数据，请先对 CPU 卡片复位后，选择相应二进制文件
unsigned int offset = 0;          //定义读取数据地址的偏移量变量，并赋初值
unsigned int len;             //定义读取数据长度的变量
```

```
unsigned char data[256];    //定义读取数据变量
len = 3;            //读取数据的长度为 3 字节
sw = ICC_read_file (fDev, offset, len, data);   //读取二进制文件 3 字节数据操作
if(sw != 0x9000)        //判断是否执行成功
{
    …  //操作失败后，用户的处理
}
```

2、存储卡读写函数

此部分为支持存储卡操作的函数，请确保 NAD 切换为 12。存储卡应用开发注意事项如下：

1．各种存储卡性能特点不同，在开发以存储卡为基础的应用系统时，应选取合适的存储卡，并注意本型号的读写器是否支持这种存储卡

2．为了正确读写存储卡，要求在操作卡片时，必须先对存储卡进行复位

3．CRW-V 读写器能自动识别支持的各类存储卡，但其中 AT24C32/64 卡片不能自动识别，需要复位后首先设置卡类型为 0x13，再进行正常操作

4．存储卡读写函数返回状态的意义

0x9000 操作成功
0x6282 offset+len 超出卡片容量错误
0x63Cx 校验密码错误，x 为剩余可校验次数
0x6581 写操作时，检测到失败
0x6983 卡片已锁死
0x6A86 长度 len 错误
0x6B00 偏移量 offset 错误
0x6D00 无效的操作，该卡不支持指令
0x6FF0 操作失败，卡片不识别
0x9001 卡片不合法
0x9002 MEM_AT1608_Aut()函数中外部认证没有启用或认证密钥不对时的返回值

1）存储卡复位

```
unsigned WINAPI MEM_Reset(
    HANDLE fd,
    unsigned char *len,
    unsigned char *resp
    );
```

参数：

fd   ： 已打开的端口描述符
lenr   ： 为对 IC 卡复位所返回的复位数据的长度
resp   ： 复位的数据结果

说明：

1．该函数会把当前的 NAD 值设为 0x12

2．AT24C32/64 不能自动识别,读写 AT24C32/64 先复位,然后设置卡类型为 0x13

示例：

```
//此示例用于存储卡的复位，同样可用于 CPU 卡的复位
unsigned char len;      //定义返回数据长度变量
unsigned char resp[32]  //定义复位信息数据变量
unsigned int sw;        //定义返回状态信息变量
sw = MEM_Reset(fDev, &len, resp);   //进行存储卡的复位操作
if(sw != 0x9000)        //判断是否执行成功
{
    …  //操作失败后，用户的处理
```

```
}
```

2）获取存储卡类型

```
unsigned WINAPI MEM_GetType(
    HANDLE fd,
    unsigned char *Type
    );
```

参数：

fd　　：　已打开的端口描述符

Type　：　返回的卡片类型代码

说明：

1．本函数识别大多数常用常见的存储卡

2．AT24C32/63 不能自动识别，读写 AT24C32/64 卡片需要先复位，然后设置卡类型为 0x13

3．本函数识别大多数常用常见的存储卡，返回的卡片类型代码说明

Type	卡片类型代码
08	AT24C01/02/04/08/16
09	SLE4418/28
0A	SLE4432/42
0F	AT102
12	AT1604
13	AT24C32/64
14	
15	AT88SC1608
16	AT88SC153
00	CPU 卡
FF	未识别卡

示例：

```
//此示例用于获取存储卡的类型，请注意，此函数需要在卡片正确复位后使用
unsigned char type;     //设置卡片类型变量
unsigned int sw;        //设置返回状态变量
sw = MEM_GetType(fDev, &type);  //取卡类型操作，卡片类型保存在 type 变量中
if(sw != 0x9000)        //判断是否执行成功
{
    …  //操作失败后，用户的处理
}
```

3）设置存储卡类型

```
unsigned WINAPI MEM_SetType(
```

```
    HANDLE fd,
    unsigned char Type
    );
```

参数：

```
    fd      :   已打开的端口描述符
    Type    :   欲设置的卡片类型代码
```

说明：

1．当有读写器不能正确识别，又属于读写器支持的卡类型的存储卡时，先调用复位函数，然后再按实际卡片类型设置

2．存储卡中 AT24C32/64 卡片不能自动识别，需要在复位后设置卡类型为 13，再进行读写操作

示例：

```
    //此示例用于设置 AT24C32/64 的卡片类型，需要在复位以后设置
    unsigned int sw;
    sw = MEM_SetType(fDev, 0x13);   //设置卡片为 0x13 型，以对其进行正确操作
    if(sw != 0x9000)          //判断是否执行成功
    {
        …  //操作失败后，用户的处理
    }
```

4）读存储卡

```
unsigned WINAPI MEM_read_bin(
    HANDLE fd,
    long offset,
    unsigned int len,
    unsigned char *resp
    );
```

参数：

```
    fd      :   已打开的端口描述符
    offset  :   存储卡的绝对地址
    len     :   欲读卡上的数据长度
    data    :   读出的数据
```

返回值：

```
    0xffff  通迅失败（发送命令或接收返回的数据失败）
    其它为从卡上返回的状态 SW1 SW2
```

说明：

示例：

```
    //此示例用于读取存储卡，请在对存储卡进行复位的必要的设置卡类型或验证密码后进行
    unsigned int sw;    //定义返回状态变量
    long offset;        //定义存储位置的偏移量
    unsigned int len;   //定义要读取的数据长度变量
    unsigned char resp[256];    //定义返回数据变量
    offset = 0x10;      //所需读出数据的偏移量地址赋值
    len = 0x10;         //所需读出数据的长度赋值
    sw = MEM_read_bin(fDev, offset, len, resp);     //读取 0x10 偏移量处的 0x10 个数据
    if(sw != 0x9000)          //判断是否执行成功
    {
        …  //操作失败后，用户的处理
    }
```

5）带保护位读存储卡（SLE4432/42/18/28）或者读熔丝（AT88SC102/1604/153/1608）

```
unsigned WINAPI MEM_read_bin_p(
```

```
    HANDLE fd,
    unsigned int offset,
    unsigned int len,
    unsigned char *resp
    );
```

参数：

```
    fd      :   已打开的端口描述符
    offset  :   存储卡的绝对地址（熔丝时为熔丝绝对位地址）
    len     :   欲读卡上的数据长度
    data    :   读出的数据
```

返回值：

```
    0xffff  通迅失败（发送命令或接收返回的数据失败）
    其它为从卡上返回的状态 SW1 SW2
```

说明：

```
    1．带保护位读存储卡，只适用 SLE4418/28，SLE4432/42
    2．带保护位读存储卡时，读出的数据，每个数据字节之后跟一个字节保护位说明
       01 00 02 00 03 00 04 01 05 01 06 01 07 01 08 01
       其中第 1 2 3 字节表示已写保护，第 4 5 6 7 8 字节表示未写保护
    3．SLE4442 只有前 32 字节的保护位，SLE4428 有 1024 个保护位
    4．读熔丝，只适用于 AT88SC102/1604/1608/153
    5．对于 AT88SC153/1608，offset 参数也无意义，读取的熔丝字节数值对应的含义
          07：FAB 1   CMA 1   PER 1
          06：FAB 0   CMA 1   PER 1
          04：FAB 0   CMA 0   PER 1
          00：FAB 0   CMA 0   PER 0
```

示例：

```
    //此示例用于读取带保护位的存储卡，请先复位和验证密码后再进行此操作
    unsigned int sw;    //定义返回状态变量
    long offset;        //定义存储位置的偏移量
    unsigned int len;   //定义要读取的数据长度变量
    unsigned char resp[256];    //定义返回数据变量
    offset = 0x10;      //所需读出数据的偏移量地址赋值
    len = 0x10;         //所需读出数据的长度赋值
    sw = MEM_read_bin_p(fDev, offset, len, resp);   //返回值的每个数据后跟一保护位
    if(sw != 0x9000)        //判断是否执行成功
    {
        …  //操作失败后，用户的处理
    }
```

6）按位读存储卡（SLE4402/4404/4406/4403 AT88SC102/1604）

```
unsigned WINAPI MEM_read_bit(
    HANDLE fd,
    unsigned int offset,
    unsigned int len,
    unsigned char *resp
    );
```

参数：

```
    fd      :   已打开的端口描述符
    offset  :   存储卡的绝对位地址
    len     :   欲读卡上的数据长度
```

data　　：　读出的数据

返回值：

0xffff　通迅失败（发送命令或接收返回的数据失败）

其它为从卡上返回的状态 SW1 SW2

示例：

```
//此示例用于按位读存储卡中的数据
unsigned int sw;    //定义返回状态变量
long offset;        //定义存储位置的偏移量
unsigned int len;   //定义要读取的数据长度变量
unsigned char resp[256];    //定义返回数据变量
offset = 0x10;      //所需读出数据的偏移量地址赋值
len = 0x10;         //所需读出数据的长度赋值
sw = MEM_read_bit(fDev, offset, len, resp); //返回值为对应偏移量地址的位值（00 或 01）
if(sw != 0x9000)        //判断是否执行成功
{
    … //操作失败后，用户的处理
}
```

7）写存储卡

```
unsigned WINAPI MEM_write_bin(
    HANDLE fd,
    long offset,
    unsigned int len,
    unsigned char *data
    );
```

参数：

fd　　：　已打开的端口描述符

offset　：　存储卡的绝对地址

len　　：　欲写数据的长度

data　　：　欲写的数据

返回值：

0xffff　通迅失败（发送命令或接收返回的数据失败）

其它为从卡上返回的状态 SW1 SW2

说明：

1．有效数据 data 少于 len 时，不足部分将会写入不可预测的数据

2．当 offset+len 大于卡片实际容量时，执行写操作，可能会出现不可预见的现象，应用开发时，应注意检查写入的数据地址是否超过卡片容量的最大地址

3．对每一种存储卡，在某一区域执行写操作，其含义可能不一样，请开发者参考相应卡片的技术资料

示例：

```
//此示例用于写存储卡操作，请保证在复位和必要的卡类型设置及验证密码操作后进行
long offset;        //定义写入数据的偏移量变量
unsigned int len;   //定义写入数据长度变量
unsigned char data[256];    //定义写入数据变量
offset = 0x10;      //设置偏移量为 0x10
len = 5;        //定义写入数据的长度为 0x10
memcpy(data, “\x11\x22\x33\x44\x55”, 5);      //设置写入的数据为 5 个
sw = MEM_write_bin(fDev, offset, len, data)     //将上一条中的数据写入
if(sw != 0x9000)        //判断是否执行成功
{
```

```
    …  //操作失败后，用户的处理
}
```

8）带保护位写存储卡（SLE4432/42/18/28）或者写熔丝（AT88SC102/1604/153/1606）

```
unsigned WINAPI MEM_write_bin_p(
    HANDLE fd,
    unsigned int offset,
    unsigned int len,
    unsigned char *data
    );
```

参数：

fd　　：　已打开的端口描述符

offset　：　存储卡的绝对地址（熔丝时为熔丝的位地址）

len　　：　欲写数据的长度（熔丝时无意义）

data　　：　欲写的数据（熔丝时无意义）

返回值：

0xffff　通迅失败（发送命令或接收返回的数据失败）

其它为从卡上返回的状态 SW1 SW2

说明：

1．用此函数写入卡的数据以后无法被更改（SLE4432/42/18/28）

2．带保护位写数据，data 的由数据+保护位（0x00 或 0x01）组成，0x00 表示该字节的保护位置 0，0x01 表示该字节的保护位不变，如 12003400560178019a00，为五个数据+五个保护位

3．len 写入数据的字节数，不含保护位

4．对 SLE4418/28，SLE4432/4442 是写保护位

5．对 AT88SC102/1604/153/1608 是写熔丝操作

6．对 AT88SC153/1608，offset 也无意义，熔丝按 FAB\CMA\PER 的顺序逐次烧写

7．本函数不对熔丝结果作校验

示例：

```
//此示例用于带保护位写存储卡操作，请保证在复位和必要的卡类型设置及验证密码操作后进行
long offset;        //定义写入数据的偏移量变量
unsigned int len;   //定义写入数据长度变量
unsigned char data[256];    //定义写入数据变量
offset = 0x10;      //设置偏移量为 0x10
len = 3;        //定义写入数据的长度为 3 （不带保护位）
memcpy(data, “\x11\x00\x22\x01\x33\x00”, 6);  //设置写入的 3 个数据，带 3 个保护位
sw = MEM_write_bin_p(fDev, offset, len, data)       //将上一条中的数据和保护位写入
if(sw != 0x9000)        //判断是否执行成功
{
    …  //操作失败后，用户的处理
}
```

9）按位写存储卡（AT88SC102/1604）

```
unsigned WINAPI MEM_write_bit(
    HANDLE fd,
    unsigned int offset,
    unsigned int len,
    unsigned char *data
    );
```

参数：

fd　　：　已打开的端口描述符

offset　：　存储卡的绝对位地址

len　　：　欲写入数据的长度

data　：　写入的数据，由 0x00 和 0x01 组成,表示每位的逻辑电平

返回值：

0xffff　通迅失败（发送命令或接收返回的数据失败）

其它为从卡上返回的状态 SW1 SW2

说明：

１．有多分区的卡片，只要各区都有擦写的权限，可以连续写入，地址按分区的序号连续排列

２．按位写须在按位擦除后适用

３．有效数据 data 少于 len 时，不足部分将会写入不可预测的数据

示例：

```
//此示例用于按位写存储卡操作，请保证在复位和必要的卡类型设置及验证密码操作后进行
long offset;        //定义写入数据的偏移量变量
unsigned int len;   //定义写入数据长度变量
unsigned char data[256];    //定义写入数据变量
offset = 0x10;      //设置偏移量为 0x10
len = 5;        //定义写入数据的长度为 5
memcpy(data, “\x01\x00\x01\x00\x01”, 5);  //设置写入的 5 个位数据
sw = MEM_write_bit(fDev, offset, len, data)     //将上一条中的位数据写入
if(sw != 0x9000)        //判断是否执行成功
{
    … //操作失败后，用户的处理
}
```

１０）位擦除（AT88SC102/1604）

```
unsigned WINAPI MEM_erase_bit(
    HANDLE fd,
    unsigned int offset
    );
```

参数：

fd　　：　已打开的端口描述符

offset　：　位地址，擦除该位所在区域，如果该卡是按字节擦除，则擦除位所在的整个字节

示例：

```
//此示例用于位擦除操作
unsigned int offset;        //定义所要擦除位的偏移量
unsigned int sw;        //定义函数返回状态
offset = 0x64;      //擦除位变量赋值
sw = MEM_erase_bit(fDev, offset);   //擦除 0x64 字节处的位，若为字节擦除，则擦除第九字节
if(sw != 0x9000)        //判断是否执行成功
{
    … //操作失败后，用户的处理
}
```

１１）验证 PIN（逻辑加密卡）

```
unsigned WINAPI MEM_verify(
    HANDLE fd,
    unsigned char len,
    unsigned char *pin
    );
```

参数：

fd　　：　已打开的端口描述符

len　　：　口令长度

pin　　：　口令

说明：

1．len 密码长度，如该卡有多个密码，则 len 分为高低两个半字节，高半字节表示密码标号，低半字节表示密码长度，高半字节为 0 的是用户密码或者传输密码，以下的 len 为十六进制

2．AT88SC102 的密码与 len 的对应关系如下：

len	密码
02	主密码
16	1 区擦除密码
24	2 区擦除密码

3．对 AT153 其密码组与 len 的对应关系如下：

区	写密码	读密码
1	13	23
2	33	43
传输密码		03

4．对 AT1604 只支持等分区结构的卡片，其密码组与 len 的对应关系如下：

密码组	密码	擦除
主密码	02	
1 区	12	22
2 区	32	42
3 区	52	62
4 区	72	82

5．对 AT1608 其密码组与 len 的对应关系如下：

分区	写密码	读密码
0	03	83
1	13	93

2	23	A3
3	33	B3
4	43	C3
5	53	D3
6	63	E3
7	73	F3
传输密码		73

示例：

```
//此示例为验证 AT1608 卡片的密码
unsigned char pinlen;       //定义密码长度变量
unsigned char pin[3];   //定义密码变量
pinlen = 3;     //密码长度赋值
memcpy(pin, "\x34\x34\x34", 3); //AT1608 卡片的密码赋值
sw = MEM_verify(fDev, pinlen, pin);     //验证卡片密码
if(sw != 0x9000)        //判断是否执行成功
{
    … //操作失败后，用户的处理
}
```

1 2）核对 PIN 并修改 PIN（逻辑加密卡）

```
unsigned WINAPI MEM_change_pin(
    HANDLE fd,
    unsigned char pin_len,
    unsigned char *oldpin,
    unsigned char *newpin
    );
```

参数：

fd ： 已打开的端口描述符

len ： 口令长度

oldpin ： 旧口令

newpin ： 新口令

说明：

只能修改用户密码或者传输密码，要修改多密码卡的其它密码，在校验相应密码，取得擦写权限后，使用 MEM_write_bin 函数在相应地址区写入

示例：

```
//此示例为验证并修改 AT1608 卡片的密码
unsigned char pinlen;       //定义密码长度变量
unsigned char oldpin[3], newpin[3]; //定义新旧密码变量
pinlen = 3;     //密码长度赋值
memcpy(oldpin, "\x34\x34\x34", 3);  //AT1608 卡片的现有密码赋值
memcpy(newpin, "\x56\x56\x56", 3);  //AT1608 卡片要修改的新密码赋值
sw = MEM_change_pin(fDev, pinlen, oldpin, newpin);      //验证卡片密码
if(sw != 0x9000)        //判断是否执行成功
```

```
    {
        …  //操作失败后，用户的处理
    }
```

1 3）AT88SC1608 专用认证函数

```
int WINAPI MEM_AT1608_Auth(
    HANDLE fd,
    unsigned char Q0[8],
    unsigned char Gc[8]
    );
```

参数：

fd　　：　已打开的端口描述符

Q0[8]　：　8 字节的随机数数组

Gc[8]　：　8 字节的认证密钥，发卡前可以从配置区的 0x30（绝对地址为 0x0830）处读出，发卡后由发卡商提供

示例：

```
    //此示例用于 AT1608 卡片的认证操作
    unsigned int sw;        //定义返回状态变量
    unsigned char Q0[8];        //定义随机数数据数组
    unsigned char Gc[8];        //定义认证密钥数据
    memcpy(Q0, “\x01\x02\x03\x04\x05\x06\x07\x08”, 8);    //8 字节的随机数组，可任意赋值
    Gc = ……        //8 字节的认证密钥，可由配置区读出或发卡商提供
    sw = MEM_AT1608_Auth(fDev, Q0, Gc);     //对 AT1608 卡片进行认证
    if(sw != 0x9000)        //判断是否执行成功
    {
        …  //操作失败后，用户的处理
    }
```

1 4）AT88SC153 专用认证函数

```
int WINAPI MEM_AT153_Auth(
    HANDLE fd,
    unsigned char Q0[8],
    unsigned char Gc[8]
    );
```

参数：

fd　　：　已打开的端口描述符

Q0[8]　：　8 字节的随机数数组

Gc[8]　：　8 字节的认证密钥，发卡前可以从配置区的 0x28（绝对地址为 0xE8）处读出，发卡后由发卡商提供

示例：

同 AT1608 的认证

3、常用算法及辅助函数

1）十六进制字符串转化为二进制数

```
unsigned char * WINAPI CHexToBin(
    unsigned char *bin,
    unsigned char *asc,
    unsigned int len
    );
```

参数：

bin　　：　二进制结果串：0x12，0x34，0xE1，0xFA

asc : 十六进制字符串，如“1234E1FA”
len : 十六进制字符串长度

返回值：
二进制结果串的指针

2）二进制数转化为十六进制字符串

```
unsigned char * WINAPI BinToCHex(
    unsigned char *asc,
    unsigned char *bin,
    unsigned int len
    );
```

参数：
asc : 十六进制字符串，如“1234E1FA”
bin : 二进制结果串：0x12，0x34，0xE1，0xFA
len : 二进制串长

返回值：
十六进制字符串的指针

3）哈什摘要

```
void WINAPI SHA1(
    unsigned char *sour,
    unsigned len,
    unsigned char *digest
    );
```

参数：
len : 要压缩的原文长度
sour : 要压缩的原文
digest : 20字节的摘要数据

4）DES加密

```
unsigned WINAPI SingleDES(char DESType,
    unsigned char * SingleDESKey,
    unsigned int  SourDataLen,
    unsigned char *SourData,
    unsigned char *DestData
    );
```

参数:
DESType : =1 加密
=2 解密
SingleDESKey: 8字节密钥
SourDataLen : 源数据长度
SourData : 源数据
DestData : 目标数据

返回值：
目标数据的长度

说明:
加密时，当明文长度不时8的倍数时，该函数在明文数据的后面加上16进制数字串“80 00 00...”，使其为8的倍数后加密

5）3DES加密

```
unsigned WINAPI TripleDES(
    char DESType,
    unsigned char * TripleDESKey,
```

```
    unsigned int  SourDataLen,
    unsigned char *SourData,
    unsigned char *DestData
    );
```

参数：

```
    DESType     :   =1 加密
                    =2 解密
    TripleDESKey:   16字节密钥K1K2
    SourDataLen :   源数据长度
    SourData    :   源数据
    DestData    :   目标数据
```

返回值：

目标数据的长度

说明：

加密时，当明文长度不是8的倍数时，该函数在明文数据的后面加上16进制数字串“80 00 00...”，使其为8的倍数后加密

加解密过程如下:

```
DES3-E({K1,K2},P)=E(K1,D(K2,E(K1,P)))
DES3-D({K1,K2},C)=D(K1,E(K2,D(K1,P)))
```

6）DES认证码

```
unsigned WINAPI SingleMAC(
    unsigned char * SingleMACKey,
    unsigned char *InitData,
    unsigned int  SourDataLen,
    unsigned char *SourData,
    unsigned char *MACData
    );
```

参数：

```
    SingleMACKey:   8字节密钥
    InitData    :   8字节的初始值
    SourDataLen :   用来产生mac码的原文长度
    SourData    :   用来产生mac码的原文
    MactData    :   计算出的认证码
```

mac码的计算方法参见有关标准

返回值：

认证码的长度为8

7）3DES认证码

```
unsigned WINAPI TripleMAC(
    unsigned char * SingleMACKey,
    unsigned char *InitData,
    unsigned int  SourDataLen,
    unsigned char *SourData,
    unsigned char *MACData
    );
```

参数：

```
    SingleMACKey:   16字节密钥
    InitData    :   8字节的初始值
    SourDataLen :   用来产生mac码的原文长度
    SourData    :   用来产生mac码的原文
```

MactData ： 计算出的认证码

返回值：

认证码的长度为8

八、特殊指令说明

为了尽可能满足用户的需求，CRW-V plus 读写器设置了一些特殊的指令，用于设置串口波特率、对卡片的自动复位，以及 PC/SC 模式下对多卡座的支持和特定存储卡的认证支持。

请注意，以下指令需要预先设置 NAD 为 0x12，请严格按照所列的步骤进行操作，否则可能会发生不可预知的错误！

1、读写器与计算机串口波特率的设置

CRW-V plus 系列串口读写器与计算机之间的通讯速率支持 9600bps 和 38400bps，用户可根据需要选择通讯速率，具体的设置方法如下：

1）801700430170 为设置 9600 波特率的指令，801700430117 为设置 38400 波特率的指令，发送相应的指令后，波特率的修改立即生效，此时读写器以新的速率发送返回值，因此收到乱码是正常现象

2）如果使用动态库，则首先关闭端口，然后以新的速率打开端口即可

也可以直接在 ICC_tsi_api()函数中，发送 0016 00/02 0000 指令修改相应波特率，则端口的速率直接修改，不必重新打开端口。

此方式对读写器波特率的修改仅对本次上电有效。

2、设置读写器为不自动复位

CRW-V plus 读写器出厂时默认状态为自动复位，因此插入卡片后正常状态为绿灯常亮，如果绿灯闪烁，可能是卡片未插好，请拔出后重试。用户可以设置读写器不自动复位模式，方法如下：

1）发送 8017004201 指令，读出一字节数据

2）将读出的数据的最高位设置为 0（可采用与 0x7F 进行“按位逻辑与”运算的方法）

3）发送 8017004201 DATA（第 2 步中得到的数据），设置成功

设置完毕后，对卡片操作时，必须先对卡复位。此设置仅对本次上电有效，如果需要恢复自动复位模式，只需要按上面的步骤，在第 2 步中，将数据最高位置 1 即可，或者对读写器重新上电亦可。CRW-V plus 读写器的 SAM 卡座不支持自动复位。

3、设置读写器为不自动取 T=0 响应数据

如果 T=0 的卡片需要返回的数据大于等于 256 字节时（如生成 1024 位密钥对），卡片会分两次返回数据，此种情况下，需要关闭 T=0 卡片的自动取响应功能，方法如下：

1）发送 8017004201 指令，读出一字节数据

2）将读出的数据的次高位设置为 0（可采用与 0xBF 进行“按位逻辑与”运算的方法）

3）发送 8017004201 DATA（第 2 步中得到的数据），设置成功

可在返回数据大于 256 字节的命令前关闭此功能，接收成功后可以再开启自动取响应的功能，设置的方法同上，只需要在第 2 步中，将数据的次高位置 1 即可。

4、存储卡和 CPU 卡共同使用时的 I/O 切换问题

当主卡座插入存储卡，对 SAM 卡座中的 CPU 卡复位可能不成功，可以通过以下的方式解决：

1）发送 8017000301 指令，读出一字节数据

2）将得到的数据的最低位置 1（可采用与 0x01 进行“按位逻辑或”运算的方法）

3）发送 8017000301 DATA（第 2 步中得到的数据），设置成功

执行成功后，即可对 SAM 卡座的 CPU 卡进行正常的复位操作，会存在此问题的存储卡类型主要是 SPI 和 I^2C 总线的卡，包括 AT24C01/02/04/08/16/32/64 卡。

5、主卡座电压切换指令

CRW-V plus 读写器的主卡座支持对 5V、3V 和 1.8V 的卡以相应的电压进行操作，操作前需要进行电压切换，复位后将以设置的电压对卡片操作，设置步骤如下：

1）发送 8017004701 指令，读出一字节数据

2）将得到数据的高两位进行设置，00 对应无电压输出，01 对应 1.8V，10 对应 3V，11 对应 5V，得到相应的值

3）发送 8017004701 DATA（第 2 步中得到的数据），设置成功

说明：默认的主卡座电压为 5V

6、PC/SC 模式的特殊指令

CRW-V plus 读写器的 PC/SC 功能，支持对读写器的主卡和 SAM 卡操作，同时为了方便用户使用，可以在此模式下，通过指令对 AT1608 和 AT153 卡进行外部认证。指令格式如下：

00 15 P1 P2 Len Data

参数：

1）P1=0x80，为设置 Nad 的命令，对各个卡座的操作可以通过以下指令实现：

0015800000 指令执行成功后，后续的指令是对主卡座的 CPU 卡进行操作

0015801200 指令执行成功后，后续的指令是对主卡座的 CPU 卡和 MEM 卡以及读写器进行操作

0015801300 指令执行成功后，后续的指令为对 SAM 卡座的 CPU 卡进行操作

2）P1=0x81，为设置 I/O 状态命令

0015810000 设置 I/O 状态为输入模式

0015810100 设置 I/O 状态为输出模式

此指令主要是针对第 4 项的 I/O 切换问题进行的封装,用户只需要发送 0015810000 即可实现以上的各步操作

3）P1=0x82，为存储卡的外部认证命令。P2=0x15 时对应 AT1608 卡，P2=0x16 时对应 AT153 卡，此时 Data 为 Q0[8]和 Gc[8]。相关的说明请参考动态库函数说明一节中第二小节的第 13 和 14 的专用认证函数

附录　PC/SC 简介

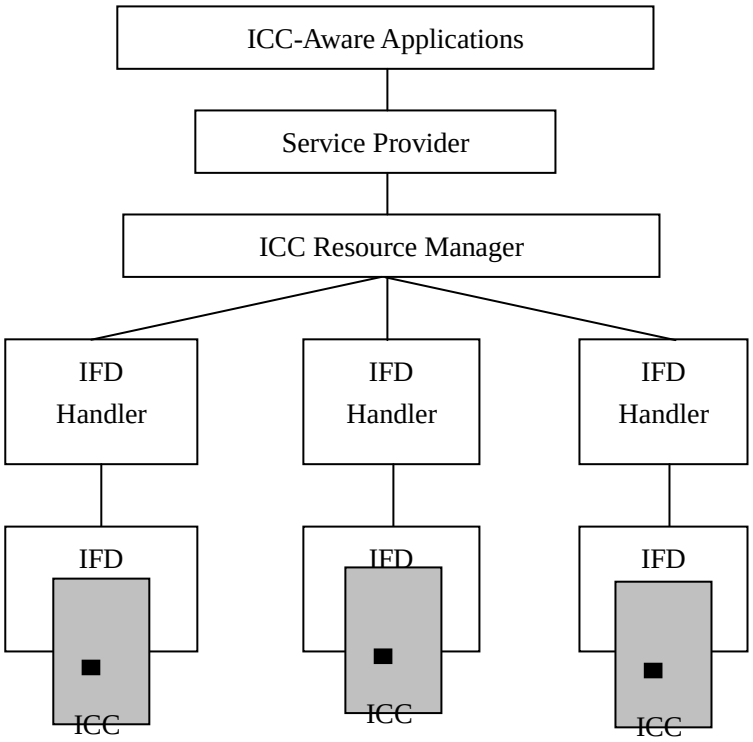
PC/SC 规范由微软公司与世界其它著名的智能卡厂商组成的 PC/SC 工作组提出的。PC/SC 规范是一个基于 WINDOWS 平台的一个标准用户接口(API),提供了一个从个人电脑(Personal Computer)到智能卡(SmartCard)的整合环境，虽然到目前为止，WONDOWS 是唯一支持 PC/SC 标准的操作系统平台，但由于 WINDOWS 的影响力，PC/SC 规范也为智能卡业界所接收。到目前为止，PC/SC 规范的最新版本是 PC/SC Specifications 2.0。

PC/SC 规范建立在工业标准-ISO7816 和 EMV 标准的基础上，但它对底层的设备接口和独立于设备的应用 API 接口（例如用来允许多个应用共享使用系统同一张智能卡的资源管理器）做了更详尽的补充。它的提出主要是为了达到以下目标：

- 遵从现在 ICC 和 PC 的标准并在适当的地方予在扩充
- 跨平台的可操作性，使该规范可在多种硬件和软件平台上实现
- 应用程序可以采用不同厂商提供的产品（独立于厂商）
- 无需要写应用代码就可以享受技术进步的好处（独立于应用）
- 建立应用级的智能卡服务接口，推广 ICC 在 PC 上的应用，并促成 PC 采用 ICC 作主标准设备。

１．PC/SC 体系的主要组成：

PC/SC 体系由三个主要部件组成，分别规定的操作系统厂商、读写器（IFD）厂商、智能卡（ICC）厂商的职责。



i.　IFD（即读写器）控制器是由 IFD 厂商提供的可安装部件。

ii. Resource manager（资源管理器）使用 Win32API 函数实现，是由操作系统厂商提供的系统级部件。

iii. Service Providers（服务提供者），服务程序是由厂商提供的可安装部件，用于提供访问特殊服务的手段，其使用的是基本 COM 的界面方式。

２．PC/SC 的 API 函数使用方法及示例

PC/SC 的 API 函数由操作系统提供，在微软公司提供的 MSDN（2000 年 10 月版）有相关帮助（路径 \\MSDN\Platform SDK\Security\Smart Card)，函数声明在 Winscard.h 中，所用的库是 Scarddlg.lib，这里只是摘录其中一个来说明：ScardConnect。

ScardConnect 函数用来建立应用程序与插在读写器中的智能卡的连接。如果在指定的读写器中没有卡片，则返回一个错误。

LONG SCardConnect(

```
IN SCARDCONTEXT hContext,
IN LPCTSTR szReader,
IN DWORD dwShareMode,
IN DWORD dwPreferredProtocols,
OUT LPSCARDHANDLE phCard,
OUT LPDWORD pdwActiveProtocol
);
```

参数：

hContext：标识资源管理器环境的句柄，这个资源管理器环境是预先调用 SCardEstablishContext 来设置的。

SzReader：与目标卡相连的读写器名字

DwShareMode：用来标识其它应用程序是否与该卡相连的标志。可以取以下值

取值	含义
SCARD_SHARE_SHARED	本应用程序将与其它应用程序共享这张卡
SCARD_SHARE_EXCLUSIVE	本应用程序将不会与其它应用程序共享这张卡
SCARD_SHARE_DIRECT	本应用程序指定读写器为私有，并直接控制卡片，其它应用程序不能访问它

DwPreferredProtocols：连接时，所能接收的协议位标志。可以取以下值，也可以进行 OR 操作

取值	含义
SCARD_PROTOCOL_T0	T=0 是可能接收的协议
SCARD_PROTOCOL_T1	T=1 是可能接收的协议
0	该参数只有当 *dwShareMode* 设置为 SCARD_SHARE_DIRECT 时，才可能为零。因此，由驱动来完成无协议的通讯直到 ScardControl 发送控制指示 IOCTL_SMARTCARD_SET_PROTOCOL

PhCard：返回一个句柄，标识智能卡与指写读写器的连接

PdwActiveProtocol：返回一个标识，标识确定有效的协议，可以取以下值

取值	含义
SCARD_PROTOCOL_T0	T=0 是有效的协议
SCARD_PROTOCOL_T1	T=1 是有效的协议
SCARD_PROTOCOL_UNDEFINED	已经指定为 SCARD_SHARE_DIRECT，因此没有协议问题，也可能读写器无卡

返回：

SCARD_S_SUCCESS ： 成功

其它 ： 错误代码，请参考错误列表

说明：

ScardConnect 是智能卡和读写器的访问函数，其它访问函数的说明，请参考相关手册

示例：

```
SCARDHANDL  hCardHandle;
LONG        lReturn;
DWORD       dwAP;

//connect to the reader
//hContext is a SCARDCONTEXT previously set by a call to ScardEstablishContext
lReturn = SCardConnect(m_hSCardContext,
    “WatchData CRW-V Plus PC/SC Reader 0”,    //打开第一个插入的 CRW-Vp 读写器
    SCARD_SHARE_SHARED,
    SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1,
    &hCardHandle,
```

```
    &dwAP);
if( SCARD_S_SUCCESS != lReturn )
{
    printf("Failed SCardConnect\n");
    goto HP;
}

// Use the connection; here we will merely display the
// active protocol.
switch ( dwAP )
{
case SCARD_PROTOCOL_T0:
    printf("Active protocol T0\n");
    break;
case SCARD_PROTOCOL_T1:
    printf("Active protocol T1\n");
    break;
case SCARD_PROTOCOL_UNDEFINED:
default:
    printf("Active protocol unnegotiated or unknown\n");
    break;
}
```

修订记录

时间	版本	修订内容
2012-5-7	1.2	针对 W1584 读卡器进行手册更新，去掉 AT45DB041 卡片的操作说明，因 CRW-VIp 系列 IC 卡读写器不支持此卡片
2007-11-1	1.1	修改稿，项目组对其中的一些错误进行修订，做为正式发布版
2007-9-10	1.0	初稿