Mifare DESFire 非接触式多应用 IC 卡开发

用户手册 V1.01

目录

1 Mifare DESFire MF3 IC D40 简介

1.1 RF 接口: ISO 14443 Type A

- 非接触数据传输, 并且无需电源供电.
- 操作距离: ≤100 mm
- 操作频率: 13.56 MHZ
- 快速数据传输: 106 kbit/s, 212 kbit/s, 424 kbit/s
- 真正的防冲突
- 7 byte 卡片唯一序列号
- 使用 ISO 14443-4 传输协议

1.2 不易丢失的内存 (NV-Memory)

- 容量为 4k 字节
- 写时间: 2 ms (1ms 擦除, 1ms 写)
- 数据可保存 10 年
- 可以循环写 100 000 次
- 灵活的文件系统
- 在一张卡上可以同时有 28 个应用
- 每个应用下可以建立 16 个文件

1.3 安全性

- 每张卡都有唯一的卡片序列号
- 3 PASS 相互认证
- RF 通道实现 DES/3DES 数据加密, 做到重放攻击保护。
- 4 字节 MAC 数据认证
- 应用级密码认证

补充说明:

卡片和读写器之间有三个安全级别的数据传输方式:

1) 明文传输
2) 明文传输+DES/3DES 加密校验(MAC)
3) DES/3DES 加密数据传输

注意：如果密钥的前 8 个字节与后 8 个字节相同，则为 DES 加密。反之，则为 3DES 加密。

每一张卡都有一个主密钥, 用于保证卡片级的安全性.

每张卡可以建28个应用, 每个应用可以有多达14个不同的用户自定义的密钥, 这些

密钥可以被分配给应用下面的文件来控制数据的访问. 一个密钥可以被多个文件共同使用。

每个应用下面可以建立多达 16 个不同大小和类型的文件, 每个文件可以定义为不同级别的访问权限. 文件的类型有下面 5 种:

- 标准数据文件
- 备份数据文件
- 带备份值操作文件
- 带备份线性记录文件
- 带备份循环记录文件

每个文件都有一个 16 字节的访问权限控制码，包括 4 种访问权限：

- ✧ 读权限 （ReadData，GetValue，Debit）
- ✧ 写权限 （WriteData Debit，LimitedCredit）
- ✧ 可读可写权限 （ReadData，WriteData，GetValue，Debit，LimitedCredit，Credit）
- ✧ 修改访问权限

每种访问权限由 4 个 bit 组成，即半个字节， 有 16 个不同的。

- 0 到 13 表示必须认证存储在对应的密钥文件中的密钥。
- 0xE 表示自由访问，不需要认证任何密钥，可以直接访问该文件。

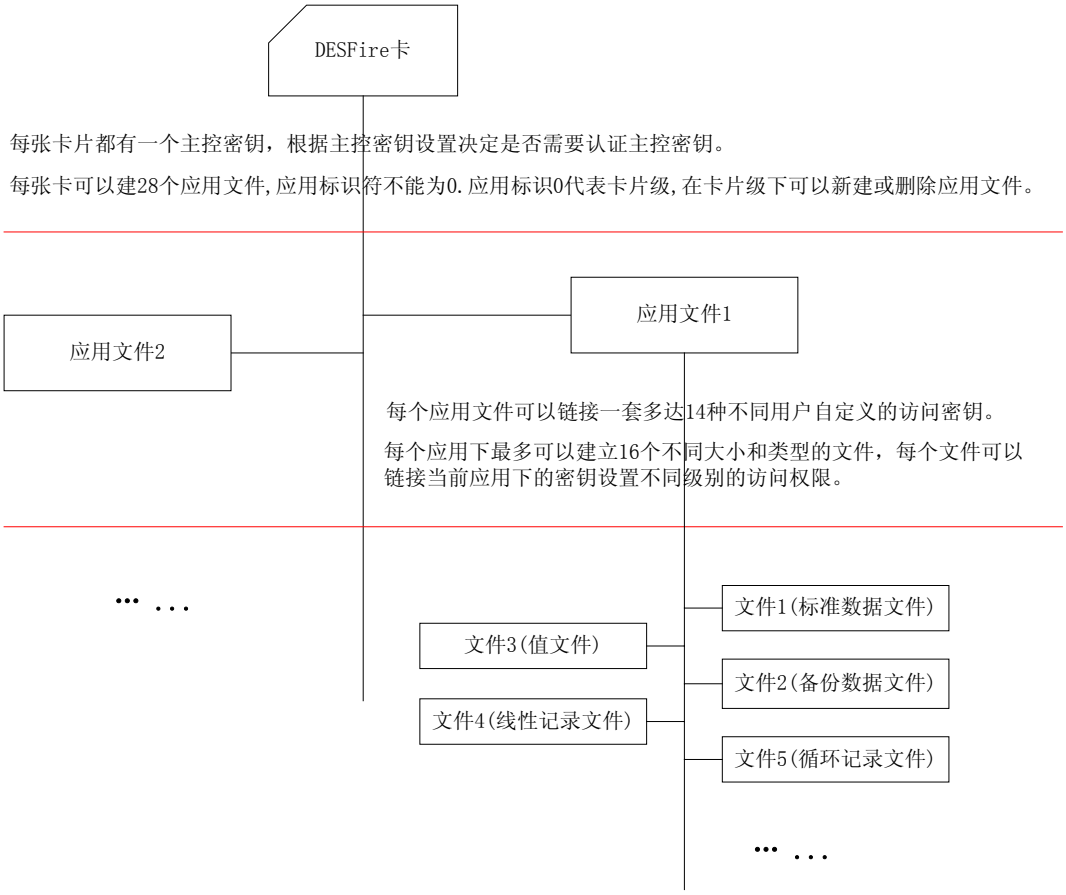
如果读权限和可读可写权限（或者写权限和可读可写权限）只有一个指定了密钥，另外一个为 0xE，如果通讯设置为 MAC 校验或者加密通讯，而且认证了指定密钥，那么通讯就为 MAC 校验或者加密通讯；没有认证，则为明文通讯。

- 0xF 表示永远不可以访问。

15 12	11 8	7 4	3 0
读权限	写权限	可读/可写权限	修改读写权限

2 DESFire 卡应用

2.1 DESFire 卡应用基本结构



2.2 应用实例

这里我们提供一个简单的 DESFire 卡的应用实例。

1. 卡片级控制

   主控密钥设置为默认值 0xff:

   - 主控密钥设置信息可以修改，但需要认证卡片主控密钥。
   - 允许不认证卡片主控密钥就可以新建应用。
     删除应用需要认证当前应用的主密钥或卡片主控密钥。
   - 读取卡片所有应用的标识符和读取卡片主控密钥设置信息不需要认证卡片主控密钥。
   - 卡片主控密钥可以修改。

卡片不管是主控密钥还是应用主密钥编号都为 0。

实现过程：

(1) 认证卡片主控密钥，主控密钥的默认值为 16 个 0x00。

(2) 建应用 1，应用标识符为 0x000001，密钥数为 5，密钥设置为 0xEF。

- 修改密码需要认证原密码。
- 应用密钥设置信息可以修改，但需要认证应用主密钥。
- 建文件/删除文件不需要认证应用主密钥。
- 读取应用所有文件的标识符和读取应用密钥设置信息不需要认证应用主密钥。
- 应用主密钥可以修改，但需要认证原应用主密钥。

修改密码：由于建应用时 5 个密码都被初始化为 16 个 0x00, 所有需要修改密码。

我们使用 8 字节密码，所以前 8 字节和后 8 字节相同。

密码 1：编号为 0x01，新密码为 0x22222222222222222222222222222222。

密码 2：编号为 0x02，新密码为 0x44444444444444444444444444444444。

密码 3：编号为 0x03，新密码为 0x66666666666666666666666666666666。

密码 4：编号为 0x04，新密码为 0x88888888888888888888888888888888。

密码 5：编号为 0x05，新密码为 0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa。

修改所有密码都必须先认证原密码。

(3) 建应用 2，应用标识符为 0x000002，密钥数为 8，密钥设置为 0x0F。

- 修改密码需要认证应用主密钥。
- 应用密钥设置信息可以修改，但需要认证应用主密钥。
- 建文件/删除文件不需要认证应用主密钥。
- 读取应用所有文件的标识符和读取应用密钥设置信息不需要认证应用主密钥。
- 应用主密钥可以修改，但需要认证原应用主密钥。

修改密码：由于建应用时 5 个密码都被初始化为 16 个 0x00, 所有需要修改密码。

我们使用 8 字节密码，所以前 8 字节和后 8 字节相同。

密码 1：编号为 0x01，新密码为 0x12121212121212121212121212121212。

密码 2：编号为 0x02，新密码为 0x14141414141414141414141414141414。

密码 3：编号为 0x03，新密码为 0x16161616161616161616161616161616。

密码 4：编号为 0x04，新密码为 0x18181818181818181818181818181818。

密码 5：编号为 0x05，新密码为 0x32323232323232323232323232323232。

密码 6：编号为 0x06，新密码为 0x20202020202020202020202020202020。

密码 7：编号为 0x07，新密码为 0x30303030303030303030303030303030。

密码 8：编号为 0x08，新密码为 0x40404040404040404040404040404040。

修改所有密码都必须先认证应用主密码。

2. 应用级控制

① 选择应用 1

② 建文件

文件 1：文件类型为标准数据文件，文件标识符为 0x00，文件大小 300，访问权限为 0x1230。

文件 2：文件类型为值文件，文件标识符为 0x01，最大值 1000，最小值 0，当

前值为 0，访问权限为 0x4550。

③ 选择应用 2

④ 建文件

文件 1：文件类型为备份数据文件，文件标识符为 0x00，文件大小 300，访问权限为 0x1230。

文件 2：文件类型为循环记录文件，文件标识符为 0x01，记录大小为 30，最大记录数为 30，访问权限为 0x4560。

3. 文件操作

① 选择应用 1

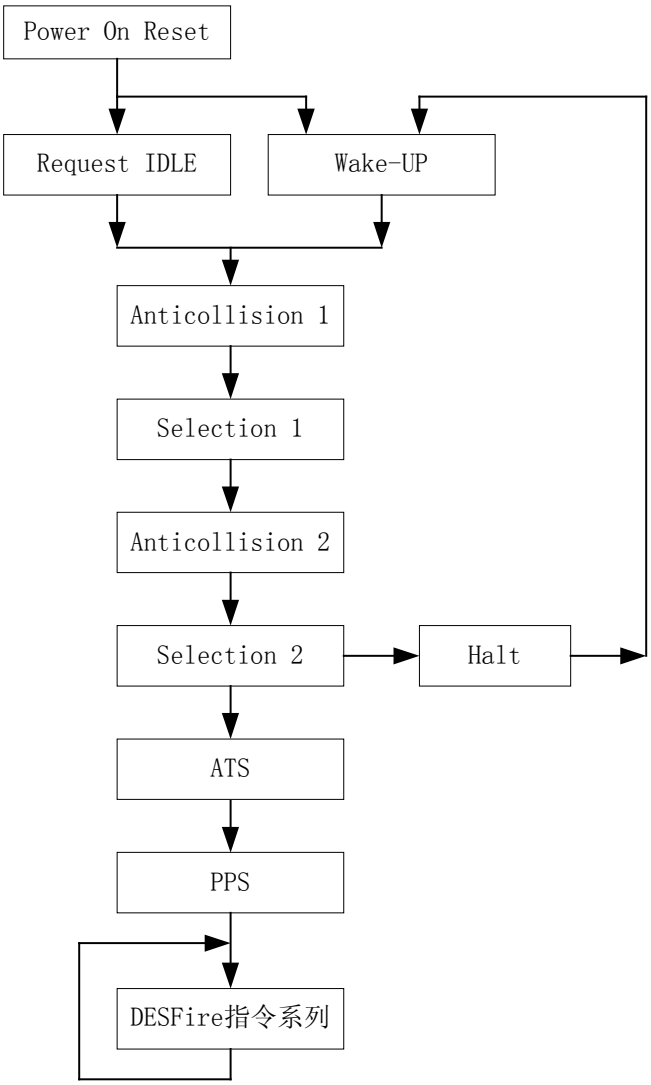
认证密码 2
写数据到文件 1
认证密码 1
读文件 1
认证密码 5
文件 2 加值 300
认证密码 4
文件 2 读当前值
文件 2 减值 30
文件 2 读当前值

② 选择应用 2

认证密码 2
写数据到文件 1
认证密码 1
读文件 1
认证密码 5
文件 2 写记录
认证密码 4
文件 2 读记录
认证密码 6
清空文件 2

3 API 函数库介绍

3.1 卡片基本操作流程



补充说明：

Request IDLE: 读写器发出 Request 指令，只有处于 IDLE 状态的卡片响应。

Wake-Up: 读写器发出 Request 指令，处于 IDLE 状态和 HALT 状态的卡片都响应。

可以调用 rf_request () 和 rf_request_std ()函数。

Anticollision 1: 第一级防冲突，只能返回某张卡的部分卡片序列号。由于 DESFire 卡有 7 个字节的卡片序列号，所以必须经过两级的防冲突和选卡才能完全确定这张卡片，并选中这张卡片。可以调用 rf_anticoll () 或者 rf_anticoll_level ()函数。

Selection 1: 第一级选卡。可以调用 rf_select ()函数。

Anticollision 2: 第二级防冲突。可以调用 rf_anticoll_level ()函数。

Selection 2: 第二级选卡。可以调用 rf_select_2 ()函数。
ATS: Answer To Select, 可以调用 rf_desfile_ats ()函数。
PPS: 协议选择，可以调用 rf_desfile_pps ()函数。

3.2 错误码列表

错误码	描述
0x00	成功
0x01	在操作区域内无卡
0x02	CRC 校验错误
0x03	数值溢出
0x05	奇偶校验错误
0x06	通讯错误
0x08	防冲突过程中读系列号错误
0x0B	从卡片接收到的位数错误
0x0C	Backup 文件没有改变，不需要 CommitTransaction 或者 AbortTransaction
0x0E	内存不足以完成该指令
0x1C	该命令码不支持
0x1E	CRC 或 MAC 码不匹配 填补的字节无效
0x40	指定的密钥号无效
0x7E	命令字符串的长度无效
0x9D	当前配置或状态不允许该请求
0x9E	参数的值无效
0xA0	请求的应用标识不存在
0xA1	在应用内不可恢复的错误，该应用将无效
0xAE	当前认证状态不允许该请求指令
0xAF	期望再发数据帧
0xBE	尝试超出文件范围读/写数据
0xC1	卡片内不可恢复的错误，卡片将无效
0xCD	卡片由于不可恢复的错误而无效
0xCE	应用的总数最大为 28，不可以再创建应用
0xDE	不可以创建该文件或应用，该文件号或应用号已存在
0xEE	因为掉电，内部备份，反转机制不能完成写操作
0xF0	指定的文件号不存在
0xF1	文件内不可恢复的错误，该文件无效

错误码	描述
-0x10	通讯错误
-0x11	通讯超时
-0x20	打开断口错误
-0x21	获得端口参数错误
-0x22	设置端口参数错误
-0x23	关闭断口出错
-0x24	端口被占用
-0x30	格式错误
-0x32	数据长度错误
-0x55	认证错误
-0x56	已冻结, 操作失败
-0x57	CRC 出错
-0x58	接收的数据的长度不对
-0x59	MAC 出错
-0x5a	永远不可访问错误

3.3 mwrf32.dll 中设备操作函数说明

在操作卡片和设备之前，必需先出始化串口。退出应用程序之前必需关闭当前正在使用的串口。

HANDLE rf_init(int port,long baud);

功 能：初始化串口

参 数：port：串口号，取值为 0～3

baud：为通讯波特率 9600～115200

返 回：成功则返回通讯设备标识符

例：HANDLE icdev;

icdev=rf_init(0,9600);

void rf_exit(HANDLE icdev);

功 能：恢复串口

参 数：icdev：通讯设备标识符

返 回：无

例：rf_exit(icdev);

int rf_get_status(HANDLE icdev,unsigned char *_Status);

功 能：取得读写器硬件版本号，如“RFMF-3.00C”

参 数：icdev：通讯设备标识符

_Status：返回读写器硬件版本信息

返 回：成功则返回 0

例：int st;

unsigned char version[11];
st=rf_get_status(icdev,version);

int rf_srd_snr(HANDLE icdev,__int16 lenth,unsigned char *rec_buffer);
功 能：取得读写器产品序列号

参 数：icdev：通讯设备标识符
length：字符串长度，其值为 16
rec_buffer：存放要读出的序列号字符串
返 回：成功则返回 0
例：st=rf_srd_snr(icdev,16,buffer);

int lib_ver(unsigned char *str_ver);
功 能：读取软件版本号,与读写器无通讯
参 数：str_ver：存放版本号的缓冲区
返 回：成功则返回 0
例：unsigned char buffer[12];
st=lib_ver(buff);

int rf_setbright(HANDLE icdev,unsigned char bright);
功 能：设置数码管显示亮度
参 数：icdev：通讯设备标识符
bright：亮度值，0～15 有效，0 表示最暗，15 表示最亮
返 回：成功则返回 0
例：st=rf_setbright(icdev,10);

int rf_ctl_mode(HANDLE icdev,unsigned char mode);
功 能：设置读写器数码管受控方式，关机后可保存设置值
参 数：icdev：通讯设备标识符
mode：受控方式
0——数码管显示受计算机控制
1——数码管显示受读写器控制（出厂设置）
显示模式由 rf_disp_mode 设置
返 回：成功则返回 0
例：st=rf_ctl_mode(icdev,0x01);

int rf_disp_mode(HANDLE icdev,unsigned char mode);
功 能：设置读写器数码管显示模式，关机后可保存设置值
参 数：icdev：通讯设备标识符
mode：显示模式
0——日期，格式为“年-月-日（yy-mm-dd）”
1——时间，格式为“时-分-秒（hh-nn-ss）”
返 回：成功则返回 0
例：st=rf_disp_mode(icdev,0x01);

int rf_disp8(HANDLE icdev,__int16 disp_len,unsigned char disp_str);

功 能：在读写器数码管上显示数字

参 数：icdev：通讯设备标识符

disp_len：显示字符串的长度，长度为 8

disp_str：要显示的数据

受读写器控制时，显示的日期/时间请参照 rf_disp_mode()中定义的格式；

受计算机控制时，显示方式由显示数据决定；每个字节的最高位为 1 表示本位数后的小数点亮，为 0 表示小数点灭。

返 回：成功则返回 0

例：int st;

```
unsigned char datebuff[8]=?9-05-20?
unsigned char numbuff1[8]={0x01,0x02,0x03,0x04,0x85,0x06,0x07,0x08};
unsigned char numbuff2[8]={0x88,0x87,0x86,0x85,0x84,0x83,0x82,0x81};
st=rf_ctl_mode(icdev,1);          //显示受读写器控制
st=rf_disp_mode(icdev,0);         //显示日期
st=rf_disp8(icdev,8,datebuff); //显示结果为“99-05-20”

st=rf_ctl_mode(icdev,0);          //显示受计算机控制

st=rf_disp8(icdev,8,numbuff1); //显示结果为“12345.678”
st=rf_disp8(icdev,8,numbuff2); //显示结果为“8.7.6.5.4.3.2.1.”
```

int rf_disp(HANDLE icdev,unsigned char pt_mode,unsigned short digit);

功 能：在读写器的数码管上显示数字（为低版本兼容函数，V3.0 及以上版本不能使用）

参 数：icdev：通讯设备标识符

pt_mode：小数点显示模式

0——小数点熄灭

1——个位后的小数点位亮

2——十位后的小数点位亮

3——百位后的小数点位亮

4——千位后的小数点位亮

digit：要显示的数

返 回：成功则返回 0

例：int st;

```
st=rf_disp(icdev,0,1234);         /*显示整数 1234*/
```

int rf_gettime(HANDLE icdev,unsigned char *time);

功 能：读取读写器日期、星期、时间

参 数：icdev：通讯设备标识符

time：返回数据，长度为 7 个字节，格式为“年、星期、月、日、时、分、秒”

返 回：成功则返回 0

例：int st;
unsigned char datetime[7];
st=rf_gettime(icdev,datetime);
//datetime 为“0x99,0x04,0x05,0x20,0x13,0x30,0x10”，
//表示 99 年星期四 5 月 20 日 13 时 30 分 10 秒

int rf_gettimehex(HANDLE icdev,char *time);
功 能：同 rf_gettime()，用十六进制表示
参 数：icdev：通讯设备标识符
time：长度为 14 个字节,均为数字
返 回：成功则返回 0
例：int st;
unsigned char datetime[14];
st=rf_gettime(icdev,datetime);
//datetime 为“99040520133010”，
//表示 99 年星期四 5 月 20 日 13 时 30 分 10 秒

int rf_settime(HANDLE icdev,unsigned char *time);
功 能：设置读写器日期、星期、时间
参 数：icdev：通讯设备标识符
time：长度为 7 个字节，格式为“年、星期、月、日、时、分、秒”
返 回：成功则返回 0
例：int st;
unsigned char datetime[7]={0x99,0x04,0x05,0x20,0x13,0x30,0x10};

st=rf_settime(icdev,datetime);

int rf_settimehex(HANDLE icdev,char *time);
功 能：同 rf_settime()，用十六进制表示
参 数：icdev：通讯设备标识符
time：长度为 14 个字节,均为数字
返 回：成功则返回 0
例：int st;
unsigned char datetime[14]=?90405201330l0?
st=rf_settime(icdev,datetime);

int rf_beep(HANDLE icdev,unsigned short _Msec);
功 能：蜂鸣
参 数：icdev：通讯设备标识符
_Msec：蜂鸣时限，单位是 10 毫秒
返 回：成功则返回 0
例：int st;
st=rf_beep(icdev,10); /*鸣叫 100 毫秒*/

int rf_srd_eeprom(HANDLE icdev,__int16 offset,__int16 length, unsigned char *rec_buffer);

功 能：读取读写器备注信息

参 数：icdev：通讯设备标识符

offset：偏移地址（0～383）

length：读取信息长度（1～384）

rec_buffer：读取到的信息

返 回：成功则返回 0

例：int st;

unsigned char buffer[100];

st=rf_srd_eeprom(icdev,0,100,buffer);

int rf_swr_eeprom(HANDLE icdev,__int16 offset,__int16 length,unsigned char *send_buffer);

功 能：向读写器备注区中写入信息

参 数：icdev：通讯设备标识符

offset：偏移地址（0～383）

length：读取信息长度（1～384）

send_buffer：要写入的信息

返 回：成功则返回 0

例：st=rf_swr_eeprom(icdev,0,8,(unsigned char *)“mwrf v3.0”);

int rf_reset(HANDLE icdev,unsigned __int16 _Msec);

功 能：射频读写模块复位

参 数：icdev：通讯设备标识符

_Msec：复位时间，0～500 毫秒有效

返 回：成功则返回 0

例：int st;

st=rf_reset(icdev,60);

3.4 通用函数

int hex_a(unsigned char *hex,char *a,unsigned char length);

功能: 将十六进制数转换为 ASCII 字符串, 例如 0x11 转换为”11”。

参数:

hex: 输入的十六进制数

a: 输出的 ASCII 字符串

length: 十六进制数的长度

返回: 成功则返回 0。

例:

```
unsigned char source[] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88};
char dest[17];
int st=0;

memset(dest, 0, 17);
hex_a(source, dest, 8); // the dest value is “1122334455667788”.
```

int a_hex(char *a,unsigned char *hex,unsigned char len);

功能: 将 ASCII 字符串转换为十六进制数，例如“12”转换为 0x12。

参数:

a : 输入的 ASCII 字符串

hex: 输出的十六进制数

length: ASCII 字符串的长度

返回: 成功则返回 0。

例:

```
char source[] = {a,0,a,1,a,2,a,3,a,4,a,5};
unsigned char dest[7];
int st = 0;
memset(dest, 0, 7);
// the dest value is {0xa0,0xa1,0xa2,0xa3,0xa4,0xa5}.
a_hex(source, dest, 12);
```

3.5 mwrf32.dll 中 DESFire 卡操作相关函数说明

int rf_request(HANDLE icdev,unsigned char _Mode,unsigned __int16 *TagType);
功 能：寻卡请求
参 数：icdev：通讯设备标识符
_Mode：寻卡模式
0---只有处在 IDLE 状态的卡片响应该请求
1—处在 IDLE 状态和 HALT 状态的卡片都可以响应该请求
Tagtype：卡类型值，0x0344 为 DESFire 卡
返 回：成功则返回 0
例：

```
#define IDLE        0
int st;
unsigned int tagtype[2];

st=rf_request(icdev,IDLE,tagtype)
```

int rf_request_std(HANDLE icdev,unsigned char _Mode,unsigned __int16 *TagType);
功 能：寻卡请求, 与 rf_request()函数的功能相同。
参 数：icdev：通讯设备标识符
_Mode：寻卡模式
0---只有处在 IDLE 状态的卡片响应该请求
1—处在 IDLE 状态和 HALT 状态的卡片都可以响应该请求
Tagtype：卡类型值，0x0344 为 DESFire 卡
返 回：成功则返回 0
例：

```
#define IDLE        0
int st;
unsigned int tagtype[2];

st=rf_request_std(icdev,IDLE,tagtype)
```

下面两个状态图分别描述了调用函数 rf_request() 和 rf_request_std() 时 REQUEST 指令的执行情况。

从图中可以看出区别在于调用 rf_request() 函数时 Request IDLE 在除 HALT 状态的任何状态下都可以执行成功，Request ALL 在任何状态下都可以成功。调用 rf_request_std()时 Request IDLE 只有在 IDLE 状态下可以成功，Request ALL 可以在 IDLE 和 HALT 状态下成功。

图 1 描述的是调用 rf_request()的情况。

图 2 描述的是调用 rf_request_std ()的情况。

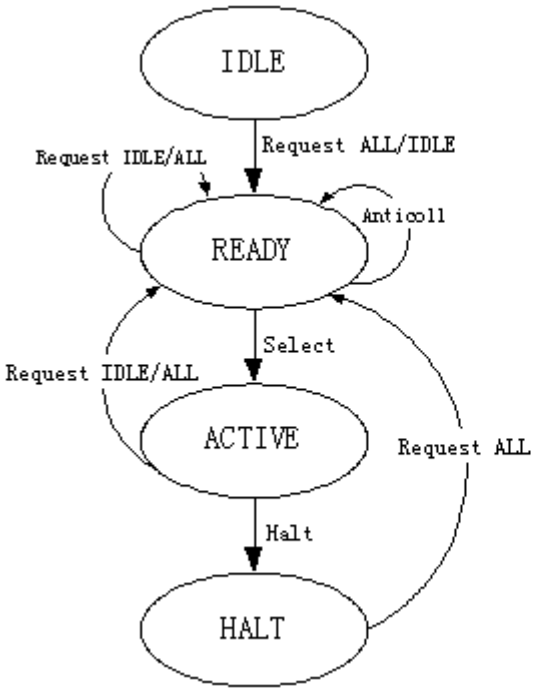


图 1

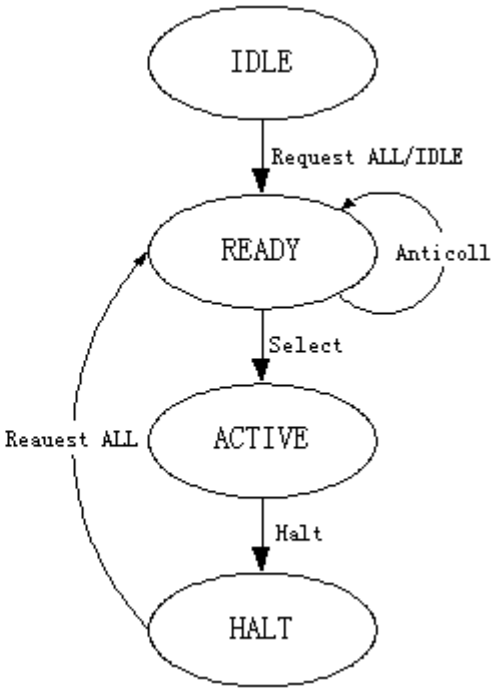


图 2

int rf_anticoll(HANDLE icdev,unsigned char _Bcnt,unsigned long *_Snr);

功 能：防止卡冲突，返回卡的序列号

参 数：icdev：通讯设备标识符

_Bcnt：预选卡所用的位数，取默认值 0。

_Snr：返回的卡序列号，对于 DESFire 卡只有前 3 个字节为序列号，第四个字节为级联标志 0x88，表示还有序列号未返回。

返 回：成功则返回 0

例：int st;

unsigned long snr;

st=rf_anticoll(icdev,0,&snr);

int rf_select(HANDLE icdev,unsigned long _Snr,unsigned char *_Size);

功 能：从多个卡中选取一个给定序列号的卡。由于 DESFire 卡的序列号是 7 个字节，所以不能完全选中这张卡。必需执行二级防冲突，即 rf_anticoll_level(2)。

参 数：icdev：通讯设备标识符

Snr：卡序列号，可以直接使用 rf_anticoll（）返回的卡片序列号的值，包括 0x88。

_Size：指向返回的卡容量的数据，该参数保留。

返 回：成功则返回 0

例：int st;

unsigned long snr;

unsigned char size[4];

st = rf_anticoll(icdev, 0, &snr);

st=rf_select(icdev,snr,size);

int rf_anticoll_level(HANDLE icdev,unsigned char level, unsigned char *_Snr);
功 能：防止卡冲突，返回卡的序列号
参 数：icdev：通讯设备标识符
level: 防冲突级别
_Snr：返回的卡序列号
对于 DESFire 卡，防冲突 1 级只有前 3 个字节为序列号的低 3 个字节，第四个字节为级联标志 0x88，表示还有序列号未返回。防冲突 2 级返回卡片序列号的高 4 个字节。
返 回：成功则返回 0
例：int st;
unsigned long snr;
st=rf_anticoll_level(icdev,2, &snr);

int rf_select_2(HANDLE icdev,unsigned char *_Snr, unsigned char *_Size);
功 能：完全地确定给定序列号的卡。
参 数：icdev：通讯设备标识符
Snr：卡序列号，可以直接使用 rf_anticoll_level（2）返回的卡片序列号的值。
_Size：指向返回的卡容量的数据，该参数保留。
返 回：成功则返回 0
例：int st;
unsigned long snr;
unsigned char size[4];

st = rf_anticoll_level(icdev, 2, &snr);
st=rf_select_2(icdev,snr,size);

int rf_desfile_ats(HANDLE icdev, unsigned char cid, unsigned char *desack);
功能： 取卡片选择应答信息。
参数： icdev：通讯设备标识符
cid：通道号
desack: 返回的应答信息
返回： 成功则返回 0
例： int st;
unsigned char desack[125];

st = rf_desfile_ats(icdev, 0, desack);

int rf_desfile_pps(HANDLE icdev, unsigned char DIV=0);
功能：协议和参数选择
参数：icdev：通讯设备标识符
DIV：卡片通讯波特率，受读写器的限制，默认为 0。
0----106 k/bit
1----212 k/bit
2----424 k/bit
返回： 成功则返回 0

例： int st;
st = rf_desfile_pps(icdev);

int rf_desfile_trn(HANDLE icdev, unsigned char cid, unsigned char *_Cmd,unsigned char _sLen, unsigned char *_Recv, unsigned char *_rLen);
功能：传送 DESFire 卡指令。
参数：icdev：通讯设备标识符
cid：通道号
_Cmd：发送的 DESFire 卡指令
_sLen：发送的 DESFire 卡指令长度
_Recv：接收的 DESFire 卡指令应答数据
_rLen：接收的 DESFire 卡指令应答数据长度
返回： 成功则返回 0
例： int st;
unsigned char Cmd[65], Recv[65];
unsigned char rlen=0;
//发送认证指令
Cmd[0] = 0x0A;
Cmd[1] = 0x00;
st = rf_desfile_trn(icdev, 0, Cmd, 2, Recv, rlen)

int rf_halt(HANDLE icdev);
功 能：中止卡操作，卡片将进入 HALT 状态。
参 数：icdev：通讯设备标识符
返 回：成功则返回 0
例：int st=0;
st=rf_halt(icdev);

如果不使用 Desfile_API.dll 函数库，用户必须调用 rf_desfile_trn 来传送 DESFire 卡片指令实现对卡片的操作；反之则不需要，只要调用 Desfile_API.dll 中的函数就可以操作。

3.6 DESfire_API.dll 中函数说明

在调用 DESFire 卡指令相关函数之前先调用 DESFireEnabled()函数打开 DESFire 接口函数库。

DESFire 卡指令系列包括安全相关指令，卡片级指令，应用级指令和数据操作指令。

3.6.1 打开 DESFire 卡接口函数库

int DESFireEnabled(HANDLE icdev);
功能：打开 DESFire 卡接口函数库
参数：icdev：通讯设备标识符
返 回：成功则返回 0
例：int st=0;
st = DESFireEnabled(icdev);

3.6.2 安全相关指令

以下介绍安全相关指令。
int Authenticate(HANDLE icdev, unsigned char cid, unsigned char _KeyNo, unsigned char *_Key)
功能：卡片与读写器三重相互认证，在认证过程中读写器和卡片都采用加密方式传递数据。该指令执行成功后将产生 16 字节的过程密钥供将来数据加密使用。
参数：icdev：通讯设备标识符
cid：通道号
_KeyNo：密钥号
_Key：认证的密钥
返 回：成功则返回 0
例：int st=0;
unsigned char key[17];
memset(key, 0,17);
//认证主密钥，密码为 16 个 0
st = Authenticate(icdev, 0, 0, key);
注意：主密钥都是由密钥号 0x00 标识。这个在卡片级和应用级都有效。

补充说明：
依赖应用的配置，执行下面操作时必须认证：

- 收集关于这个应用的信息
- 修改这个应用下的密钥

- 在这个应用内建立和删除文件
- 修改访问权限
- 访问认证了的应用下的数据文件

依赖卡片的安全配置，下面的指令可能需要认证卡片主控密钥：

- 收集关于这个卡片的应用的信息
- 修改卡片主控密钥本身
- 修改卡片密钥设置
- 建立新的应用
- 删除一个存在的应用

下列指令将改变认证状态：

- 选择一个应用
- 修改被用来达到当前有效认证状态的密钥
- 一次失败的认证

int ChangeKeySettings(HANDLE icdev, unsigned char cid, unsigned char _KeySettings);

功能：修改主控密钥/应用主密钥的密钥设置信息。指令执行过程数据采用 DES/3DES 加密和 CRC 校验。

参数：icdev：通讯设备标识符

cid：通道号

_KeySettings: 新的密钥设置信息

返 回：成功则返回 0

例：

```
int st=0;
unsigned char keyset[2];

keyset[0] = 0x0f;
st = ChangeKeySettings(icdev, 0, keyset);
```

注意：该指令只有在当前密钥设置为配置可修改时可以执行成功。

补充说明：

- 卡片主控密钥的密钥设置：

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RFU	**RFU**	**RFU**	**RFU**	**configuration changeable**	**free create/delete without PICC master key**	**free directory list access without PICC master key**	**allow changing the key**

卡片级的密钥设置的每个位说明如下：

Bit7-Bit4：保留，必须设成 0。

Bit3：编码是否允许修改卡片主控密钥设置信息：

- 0：配置再也不可以修改（冻结了）。

- 1：如果认证了卡片主控密钥，可以修改配置（默认设置）。

Bit2：编码在建立应用/删除应用时是否需要认证卡片主控密钥：

- 0：只有成功认证了卡片主控密钥才能建立/删除应用。
- 1：不认证卡片主控密钥也允许建应用（默认设置）。
  删除应用操作之前需要成功认证应用主密钥或卡片主控密钥。

Bit1：编码在应用目录访问时是否需要认证卡片主控密钥：

- 0：GetApplicationIDs 和 GetKeySettings 需要成功认证卡片主控密钥。
- 1：GetApplicationIDs 和 GetKeySettings 不需要认证卡片主控密钥（默认设置）。

Bit0：编码卡片主控密钥是否可以修改：

- 0：卡片主控密钥再也不可以修改（冻结了）。
- 1：卡片主控密钥可以修改（必须认证当前的卡片主控密钥，默认设置）。

- 应用主密钥的密钥设置

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ChangeKey Access Rights Bit3	**ChangeKey Access Rights Bit2**	**ChangeKey Access Rights Bit1**	**ChangeKey Access Rights Bit0**	**configuration changeable**	**free create/delete without master key**	**free directory list access without master key**	**allow change master key**

应用级的密钥设置的每个位说明如下：

Bit7-Bit4：控制修改应用下的所有密钥的访问权限。

- 0x0：修改应用下的任何密钥都需要认证应用主密钥（默认设置）。
- 0x1..0xD：修改应用下的任何密钥都需要认证指定密钥。
- 0xE：修改密钥需要认证与被修改的密钥相同密钥号的密钥。

Bit3：编码是否允许修改卡片主控密钥设置信息：

- 0：配置再也不可以修改（冻结了）。
- 1：如果认证了卡片主控密钥，可以修改配置（默认设置）。

Bit2：编码在建立应用/删除应用时是否需要认证卡片主控密钥：

- 0：只有成功认证了卡片主控密钥才能建立/删除应用。
- 1：不认证卡片主控密钥也允许建应用（默认设置）。
  删除应用操作之前需要成功认证应用主密钥或卡片主控密钥。

Bit1：编码在应用目录访问时是否需要认证卡片主控密钥：

- 0：GetApplicationIDs 和 GetKeySettings 需要成功认证卡片主控密钥。
- 1：GetApplicationIDs 和 GetKeySettings 不需要认证卡片主控密钥（默认设置）。

Bit0：编码卡片主控密钥是否可以修改：

- 0：卡片主控密钥再也不可以修改（冻结了）。
- 1：卡片主控密钥可以修改（必须认证当前的卡片主控密钥，默认设置）。

int GetKeySettings(HANDLE icdev, unsigned char cid,unsigned char *_KeySettings ,unsigned char *_Size);

功能：取得主控密钥/应用主密钥的当前密钥设置信息。

参数：icdev：通讯设备标识符

cid：通道号

_KeySettings: 当前的密钥设置信息

返 回：成功则返回 0

例：int st=0;

```
unsigned char keyset;

keyset[0] = 0x0f;
st = GetKeySettings(icdev, 0, &keyset);
```

int ChangeKey(HANDLE icdev, unsigned char cid,unsigned char _KeyNo, unsigned char *_OldKey, unsigned char *_NewKey);

功能：修改存储在卡片内的任何密钥。该指令执行过程中数据采用 DES/3DES 加密和 CRC 校验。

如果当前选中的 AID = 0x00, _KeyNo 必需为 0，修改的是卡片主控密钥。

如果当前选中的 AID ≠ 0x00，修改的是当前应用下的指定密钥。

密钥头 8 个字节的第 8 bit (最低位)共一个字节可以用作密钥版本信息。修改密钥也是调用该函数。如果不用作密钥版本信息，则必须确保这个字节的内容为一个默认值（如“0”），以便 GetKeyVersion 指令可以读出来。

参数：icdev：通讯设备标识符

cid：通道号

_KeyNo：密钥号

_OldKey：旧的密钥，长度为 16。

_NewKey：新的密钥，长度为 16。

返 回：成功则返回 0

例：int st=0;

```
unsigned char oldkey[17], newkey[17];
//旧的密钥为 16 个 0
memset(oldkey, 0, 17);
//新的密钥为 16 个 0x11
memset(newkey,0x11,16);
st = ChangeKey(icdev, 0, 1, oldkey, newkey);
```

补充说明：修改密码之前必须根据修改密钥的访问权限认证对应的密钥，修改密码的访问权限由密钥设置控制。

- 0x0：修改应用下的任何密钥都需要认证应用主密钥（默认设置）。
- 0x1..0xD：修改应用下的任何密钥都需要认证指定密钥。
- 0xE：修改密钥需要认证与被修改的密钥相同密钥号的密钥。

修改卡片主控密钥必须认证当前的卡片主控密钥。密钥被修改后将推出安全

状态，必须重新认证。

int GetKeyVersion(HANDLE icdev, unsigned char cid,unsigned char _KeyNo, unsigned char *_KeyVer);

功能：取得当前密钥版本信息。

如果当前选中的 AID = 0x00, _KeyNo 必需为 0，取卡片主控密钥的版本信息。

如果当前选中的 AID ≠ 0x00，取当前应用下的指定密钥的版本信息。

参数：icdev：通讯设备标识符

cid：通道号

_KeyNo：密钥号

_KeyVer: 返回的当前密钥版本信息

返 回：成功则返回 0

例：int st=0;

unsigned char keyver;

st = GetKeyVersion(icdev, 0, 1, &keyver);

补充说明：密钥头 8 个字节的第 8 bit (最低位)共一个字节用作密钥版本。调用 ChangeKey() 修改密钥版本。

3.6.3 卡片级指令

除 DeleteApplication 指令外卡片级的其他指令都要求当前选中的 AID 是 0x000000。

以下是卡片级指令：

int CreateApplication(HANDLE icdev,unsigned char cid, long _Aid, unsigned char _KeySettings,unsigned char _Number);

功能：建立新的卡片应用。

参数：icdev：通讯设备标识符

cid：通道号

_Aid：新建应用的应用标识符，3 字节数。

_KeySettings：新建应用的主密钥设置信息

_Number：新建应用的密钥数量，函数执行成功后所有的密钥都为初始值 16 个 0x00。

返 回：成功则返回 0

例：int st=0;

st = CreateApplication(icdev, 0, 1, 0xef, 5);

注意：新建的应用的密钥在最近的卡片发布的时候调用 ChangeKey()实现个人化。

根据卡片的主控密钥设置信息决定调用该函数之前是否需要认证卡片主控密钥。

int DeleteApplication(HANDLE icdev, unsigned char cid, long _Aid);

功能：删除指定的卡片应用。

参数：icdev：通讯设备标识符

cid：通道号
_Aid：被删除应用的应用标识符
返 回：成功则返回 0
例：int st=0;
st = DeleteApplication(icdev, 0, 1);

注意：根据卡片的主控密钥设置信息决定调用该函数之前是否需要认证卡片主控密钥，最近的卡都必须认证当前被删除的应用的主密钥。
执行这个函数后分配的应用标识符被删除，建立新的应用时可以使用这个删除了的应用标识符。但是被删除的内存块不可以使用，要恢复删除的内存块必须调用 FormatPICC()函数，擦除卡片整个用户内存。
即使卡片的主控密钥为默认值（全 0）并且密钥设置中为不需要认证主密钥自由建立/删除应用，认证主控密钥或应用主密钥是必须的。
如果当前选中的应用被删除，卡片自动选择卡片级，即 AID=0x000000。

int GetApplicationIDs(HANDLE icdev, unsigned char cid, unsigned char *_AidNum, long *_Aids);
功能：该函数返回卡片当前所有应用的应用标识符。
参数：icdev： 通讯设备标识符
cid： 通道号
_AidNum：应用的数量
_Aids： 返回的卡片当前所有应用的应用标识符
返 回：成功则返回 0
例：int st=0;
unsigned char aidnum=0;
long aids[30];
memset(aids, 0, 30);
st = GetApplicationIDs(icdev, 0, &aidnum, aids);

int SelectApplication(HANDLE icdev, unsigned char cid, long _Aid);
功能：选择指定的卡片应用。以后的指令将对这个应用产生作用。
参数：icdev：通讯设备标识符
cid：通道号
_Aid：被选择的应用的应用标识符
返 回：成功则返回 0
例：int st=0;
st = SelectApplication(icdev, 0, 1);

注意：执行 SelectApplication 指令将使当前认证状态无效。

int FormatPICC(HANDLE icdev, unsigned char cid);
功能：释放卡片用户内存。卡片所有应用和应用下的所有文件都被删除。
卡片的主控密钥和主控密钥设置都保持当前值，不受该指令影响。
参数：icdev：通讯设备标识符

cid：通道号
返 回：成功则返回 0
例：int st=0;
st = FormatPICC(icdev, 0);

注意：这条指令总是需要认证卡片主控密钥。

int _GetVersion(HANDLE icdev, unsigned char cid, unsigned char *_VerInfo);
功能：该函数返回卡片的制造业的相关数据
参数：icdev：通讯设备标识符
cid：通道号
_VerInfo：返回的卡片的制造业的相关数据,长度为 28。
返 回：成功则返回 0
例：int st=0;
unsigned char VerInfo[30];
memset(VerInfo, 0, 30);

st = _GetVersion(icdev, 0, 1, VerInfo);

卡片的制造业相关信息：

	0	1	2	3	4	5	6	
0	VenderID (0x04)	Type (0x01)	Subtype (0x01)	Major Version Number	Major Version Number	Memory Size	Protocol Type	硬件相关信息
1	VenderID (0x04)	Type (0x01)	Subtype (0x01)	Major Version Number	Major Version Number	Memory Size		软件相关信息
2	UID(1)	UID(2)	UID(3)	UID(4)	UID(5)	UID(6)	UID(7)	卡片序列号
3	Batch(0)	Batch(1)	Batch(2)	Batch(3)	Batch(4)	Week	year	生产批号

3.6.4 应用级指令

以下为应用级指令，这些指令要求先选择某一应用。

int GetFileIDs(HANDLE icdev,unsigned char cid, unsigned char *_FileIDNum, unsigned char *_FileIDs);

功能：取卡片当前应用目录下的所有文件的文件标识符。

参数：icdev： 通讯设备标识符

cid： 通道号

_FileIDNum：文件数量

_FileIDs： 当前应用目录下的所有文件的文件标识符

返 回：成功则返回 0

例：int st=0;

unsigned char filenum=0;

unsigned char fileIds[30];

st = GetFileIDs(icdev, 0, &filenum, fileIds);

int GetFileSettings(HANDLE icdev, unsigned char cid,unsigned char _FileNo, unsigned char *_FileSettings);

功能：取卡片当前应用目录下的指定文件的设置信息。

参数：icdev： 通讯设备标识符

cid： 通道号

_FileNo： 文件标识符

_FileSettings： 返回的文件设置信息

返 回：成功则返回 0

例：int st=0;

unsigned char fileno=1;

unsigned char fileset[20];

st = GetFileSettings(icdev, 0, fileno, fileset);

根据指定的文件的文件类型的不同，返回的数据也不同：

- 数据文件(Data File)

bytes	1	1	2	3
说明	文件类型	通讯设置	访问权限	文件大小

- 值文件(Value File)

bytes	1	1	2	4	4	4	1
说明	文件类型	通讯设置	访问权限	最小值	最大值	当前最大限存金额	是否支持限存

- 记录文件(Record File)

bytes	1	1	2	3	3	3
说明	文件类型	通讯设置	访问权限	记录大小	最大记录数	当前记录数

int ChangeFileSettings(HANDLE icdev, unsigned char cid,unsigned char _FileNo, unsigned char _ComSet, unsigned char *_AccessRights);

功能：修改卡片当前应用目录下的指定文件的设置信息。

参数：icdev： 通讯设备标识符

cid： 通道号

_FileNo： 文件标识符

_ComSet： 通讯设置，0：明文传输，1：MAC 码校验，3：DES/3DES 加密

_AccessRights：文件访问权限

_AccessRights[0]：低半字节为修改访问权限设置的权限

高半字节为可读/可写访问该文件的权限

_AccessRights[1]：低半字节为写访问该文件的权限

高半字节为读访问该文件的权限

返 回：成功则返回 0

例：
```
int st=0;
unsigned char fileno=1;
unsigned char comset=0; //明文传输
unsigned char _AccessRights[3];
//修改访问权限必须认证密码 4，认证密码 1 后可对该文件进行读、写操作。
//认证密码 2 后可以对该文件进行写操作。
//认证密码 3 后可以对该文件进行读操作。
_AccessRights[0] = 0x14;
_AccessRights[1] = 0x32;
st = ChangeFileSettings(icdev, 0, fileno, comset, _AccessRights);
```

int CreateStdDataFile(HANDLE icdev, unsigned char cid,unsigned char _FileNo, unsigned char _ComSet, unsigned char *_AccessRights, long _FileSize);

功能：在卡片当前应用目录下的建新的标准数据文件。

参数：icdev： 通讯设备标识符

cid： 通道号

_FileNo： 文件标识符，0x00 到 0x0F。

_ComSet： 通讯设置，0：明文传输，1：MAC 码校验，3：DES/3DES 加密

_AccessRights：文件访问权限

_AccessRights[0]：低半字节为修改访问权限设置的权限

高半字节为可读/可写访问该文件的权限

_AccessRights[1]：低半字节为写访问该文件的权限

高半字节为读访问该文件的权限

_FileSize： 文件大小

返 回：成功则返回 0

例：
```
int st=0;
unsigned char fileno=1;
unsigned char comset=0; //明文传输
unsigned char _AccessRights[3];
unsigned char _FileSize=0;
_AccessRights[0] = 0x14;
```

```
_AccessRights[1] = 0x32;
st = CreateStdDataFile(icdev, 0, fileno, comset, _AccessRihts, _FileSize);
```

注意：卡片给文件分配空间总是为 32 的倍数，大于或等于文件大小。

int CreateBackupDataFile(HANDLE icdev, unsigned char cid,unsigned char _FileNo, unsigned char _ComSet, unsigned char *_AccessRights, long _FileSize);

功能：在卡片当前应用目录下的建新的数据文件，支持备份机制。 卡片给该文件分配的空间总是大于或等于文件大小的两倍，且为 32 的倍数。

参数：icdev： 通讯设备标识符
cid： 通道号
_FileNo： 文件标识符，0x00 到 0x07。
_ComSet： 通讯设置，0：明文传输，1：MAC 码校验，3：DES/3DES 加密
_AccessRights：文件访问权限
_AccessRights[0]：低半字节为修改访问权限设置的权限
高半字节为可读/可写访问该文件的权限
_AccessRights[1]：低半字节为写访问该文件的权限
高半字节为读访问该文件的权限
_FileSize： 文件大小

返 回：成功则返回 0

例：

```
int st=0;
unsigned char fileno=1;
unsigned char comset=0; //明文传输
unsigned char _AccessRights[3];
unsigned char _FileSize=0;
_AccessRights[0] = 0x14;
_AccessRights[1] = 0x32;
st = CreateBackupDataFile(icdev, 0, fileno, comset, _AccessRihts, _FileSize);
```

int CreateValueFile(HANDLE icdev, unsigned char cid,unsigned char _FileNo, unsigned char _ComSet, unsigned char *_AccessRights, long _LowLimit, long _UpperLimit, long _Value, BOOL _LtCreditEnabled);

功能：在卡片当前应用目录下的建新的值文件，支持备份机制。

参数：icdev： 通讯设备标识符
cid： 通道号
_FileNo： 文件标识符，0x00 到 0x07。
_ComSet： 通讯设置，0：明文传输，1：MAC 码校验，3：DES/3DES 加密
_AccessRights：文件访问权限
_AccessRights[0]：低半字节为修改访问权限设置的权限
高半字节为可读/可写访问该文件的权限
_AccessRights[1]：低半字节为写访问该文件的权限
高半字节为读访问该文件的权限
_LowLimit： 最小值
_UpperLimit：最大值
_Value： 初始化值

_LtCreditEnabled：是否支持限存

返 回：成功则返回 0

例：int st=0;

unsigned char fileno=1;

unsigned char comset=0; //明文传输

unsigned char _AccessRights[3];

_AccessRights[0] = 0x14;

_AccessRights[1] = 0x32;

st = CreateValueFile(icdev, 0, fileno, comset, _AccessRihts, 10, 10000, 100, FALSE);

int CreateLinearRecordFile(HANDLE icdev, unsigned char cid,unsigned char _FileNo, unsigned char _ComSet, unsigned char *_AccessRights, long _RecordSize, long _MaxNum);

功能：在卡片当前应用目录下的建新的线性记录文件，支持备份机制。

参数：icdev： 通讯设备标识符

cid： 通道号

_FileNo： 文件标识符，0x00 到 0x07。

_ComSet： 通讯设置，0：明文传输，1：MAC 码校验，3：DES/3DES 加密

_AccessRights：文件访问权限

_AccessRights[0]：低半字节为修改访问权限设置的权限

高半字节为可读/可写访问该文件的权限

_AccessRights[1]：低半字节为写访问该文件的权限

高半字节为读访问该文件的权限

_RecordSize： 一条记录的大小，必须大于 0

_MaxNum： 文件最大记录数，必须大于 0

返 回：成功则返回 0

例：int st=0;

unsigned char fileno=1;

unsigned char comset=0; //明文传输

unsigned char _AccessRights[3];

_AccessRights[0] = 0x14;

_AccessRights[1] = 0x32;

st = CreateLinearRecordFile(icdev, 0, fileno, comset, _AccessRihts, 20, 50);

int CreateCyclicRecordFile(HANDLE icdev, unsigned char cid,unsigned char _FileNo, unsigned char _ComSet, unsigned char *_AccessRights, long _RecordSize, long _MaxNum);

功能：在卡片当前应用目录下的建新的循环记录文件。

参数：icdev： 通讯设备标识符

cid： 通道号

_FileNo： 文件标识符，0x00 到 0x07。

_ComSet： 通讯设置，0：明文传输，1：MAC 码校验，3：DES/3DES 加密

_AccessRights：文件访问权限

_AccessRights[0]：低半字节为修改访问权限设置的权限

高半字节为可读/可写访问该文件的权限
_AccessRights[1]：低半字节为写访问该文件的权限
高半字节为读访问该文件的权限
_RecordSize： 一条记录的大小，必须大于 0
_MaxNum： 文件最大记录数，必须大于 1
返 回：成功则返回 0
例：int st=0;
unsigned char fileno=1;
unsigned char comset=0; //明文传输
unsigned char _AccessRights[3];
_AccessRights[0] = 0x14;
_AccessRights[1] = 0x32;
st = CreateCyclicRecordFile(icdev, 0, fileno, comset, _AccessRihts, 20, 50);

int _DeleteFile(HANDLE icdev, unsigned char cid,unsigned char _FileNo);
功能：在卡片当前应用目录下的删除指定的文件。
参数：icdev： 通讯设备标识符
cid： 通道号
_FileNo： 文件标识符，0x00 到 0x0F。
返 回：成功则返回 0
例：int st=0;
unsigned char fileno=1;
st = DeleteFile(icdev, 0, fileno);

3.6.5 数据操作指令

根据文件的通讯设置，在执行数据操作指令时有明文、MAC 码校验、DES/3DES 加密等三种数据传输方式。在创建文件时指定了卡片在执行数据操作指令时的数据传输方式，也可以通过调用 ChangeFileSettings（）函数来修改文件的通讯设置。在数据操作大部分指令函数中都包含一个 mode 参数，它告诉读卡器采用何种数据传输方式。卡片和读卡器必须采用相同的数据传输方式才可保证数据操作指令的正确执行。

关于数据操作时的文件访问权限控制有下面几种情况：

1、 如果该文件对应该操作的访问权限为 0xE，即不需要认证任何密钥就可以自由地操作。数据传输方式必须为明文传输。
2、 如果该文件对应该操作的访问权限为 0x0 到 0xD, 即需要认证指定密钥后才可以执行该数据操作，0x0 为认证主控密钥。如果未认证或认证失败，则卡片强制数据传输方式为明文传输。如果认证指定密钥成功，则必须按照指定数据传输方式传输数据。
3、 如果该文件对应该操作的访问权限为 0xF，则该文件永远都不可以执行该操作。

以下是数据操作指令：

➢ **数据文件操作：读数据和写数据**

int ReadData(HANDLE icdev, unsigned char cid, unsigned char mode, unsigned char _FileNo, long _Offset, long _Length, unsigned char *_Data, unsigned long *datalen);

功能：读取卡片当前目录下指定数据文件（标准数据文件或备份数据文件）的数据。

参数：icdev： 通讯设备标识符
cid： 通道号
mode： 数据传输方式，0：明文，1：MAC 码校验，2：DES/3DES 加密
_FileNo： 文件标识符，0x00 到 0x0F，如果该文件为备份文件，则必须小于 7。
_Offset： 文件偏移地址
_Length： 读取的数据长度，如果为 0，则表示读取从 Offset 开始的全部数据
_Data： 读出的数据
_datalen： 返回的数据的长度

返 回：成功则返回 0

例：
```
int st=0;
unsigned char fileno=1;
unsigned char data[35];
unsigned long len=0;
st = ReadData(icdev, 0, fileno, 0, 30, data, &len);
```

注意：如果在写数据之后提交操作之前读取备份文件数据，则读出的数据为旧的数据。
在执行该操作之前需要认证读权限或可读/可写权限指定的密钥。

int WriteData(HANDLE icdev, unsigned char cid, unsigned char mode, unsigned char _FileNo, long _Offset, long _Length,unsigned char *_Data);

功能：向卡片当前目录下的指定数据文件（标准数据文件或备份数据文件）写数据。

参数：icdev： 通讯设备标识符
cid： 通道号
mode： 数据传输方式，0：明文，1：MAC 码校验，2：DES/3DES 加密
_FileNo： 文件标识符，0x00 到 0x0F，如果该文件为备份文件，则必须小于 7。
_Offset： 文件偏移地址
_Length： 写数据长度，必须大于 0
_Data： 写入的数据
返 回：成功则返回 0

例：
```
int st=0;
unsigned char fileno=1;
unsigned char data[35];
unsigned long len=0;
memset(data, 0x11, 30);//写入 30 个 0x11
st = WriteData(icdev, 0, fileno, 0, 30, data);
```

注意：在执行该操作之前需要认证写权限或可读/可写权限指定的密钥。
如果操作的文件为备份文件，则必须执行 CommitTransaction 指令提交操作写入的数据才会有效。也可以调用 AbortTransaction 指令取消所有修改。

➢ **值文件操作：读取当前值，存入（加值），取出（减值）和 限存（有限地加值）**

int GetValue(HANDLE icdev, unsigned char cid, unsigned char mode, unsigned char _FileNo, long *_Value);

功能：读取卡片当前目录下指定值文件的当前值。

参数：icdev： 通讯设备标识符
cid： 通道号
mode： 数据传输方式，0：明文，1：MAC 码校验，2：DES/3DES 加密
_FileNo： 文件标识符，0x00 到 0x07。
_Value： 读出的文件的当前值

返 回：成功则返回 0

例：int st=0;
unsigned char fileno=1;
long value=0;
st = GetValue(icdev, 0, 0, fileno, &value);

注意：在执行该操作之前需要认证读权限或可读/可写权限指定的密钥。
如果更新了值文件的值，但未执行 CommitTransaction 指令，则 GetValue 总是返回旧的未修改的值。

int Credit(HANDLE icdev, unsigned char cid, unsigned char mode, unsigned char _FileNo, long _Value);

功能：当前目录下指定值文件加值。

参数：icdev： 通讯设备标识符
cid： 通道号
mode： 数据传输方式，0：明文，1：MAC 码校验，2：DES/3DES 加密
_FileNo： 文件标识符，0x00 到 0x07。
_Value： 增加的值

返 回：成功则返回 0

例：int st=0;
unsigned char fileno=1;
long value=1000;
st = Credit(icdev, 0, 0, fileno, value);

注意：在执行该操作之前需要认证可读/可写权限指定的密钥。
更新的值必须在执行 CommitTransaction 指令后才有效。也可以调用 AbortTransaction 指令取消所有修改。
该指令永远不可以修改 LimitedCredit 的值。

int Debit(HANDLE icdev, unsigned char cid, unsigned char mode, unsigned char _FileNo, long _Value);

功能：当前目录下指定值文件减值。

参数：icdev： 通讯设备标识符
cid： 通道号
mode： 数据传输方式，0：明文，1：MAC 码校验，2：DES/3DES 加密

_FileNo： 文件标识符，0x00 到 0x07。
_Value： 减去的值

返 回：成功则返回 0

例：int st=0;
unsigned char fileno=1;
long value=100;
st = Debit(icdev, 0, 0, fileno, value);

注意：在执行该操作之前需要认证读权限、写权限或可读/可写权中任何一个限指定的密钥。
更新的值必须在执行 CommitTransaction 指令后才有效。也可以调用 AbortTransaction 指令取消所有修改。
如果该文件支持 LimitedCredit，Debit 操作减去的值被设置成后来的执行 CommitTransaction 指令之前的 LimitedCredit 操作的限定的最大值。

int LimitedCredit(HANDLE icdev, unsigned char cid, unsigned char mode, unsigned char _FileNo, long _Value);

功能：当前目录下指定值文件加限定的值。该操作必须在 Debit 操作后执行，而且每执行完一次 Debit 操作只能执行一次 LimitedCredit 操作，而且该操作加的值必须小于或等于 Debit 操作减去的值。

参数：icdev： 通讯设备标识符
cid： 通道号
mode： 数据传输方式，0：明文，1：MAC 码校验，2：DES/3DES 加密
_FileNo： 文件标识符，0x00 到 0x07。
_Value： 加的值

返 回：成功则返回 0

例：int st=0;
unsigned char fileno=1;
long value=50;
st = LimitedCredit(icdev, 0, 0, fileno, value);

注意：在执行该操作之前需要认证写权限或可读/可写权限指定的密钥。
更新的值必须在执行 CommitTransaction 指令后才有效。也可以调用 AbortTransaction 指令取消所有修改。
LimitedCredit 指令操作的值受最近一次 Debit 操作减去的值限制。执行完一次 LimitedCredit 操作后新的限定值为 0，也就是说不能再执行该操作直到下一个 Debit 操作执行完成。一次 Debit 操作只能使用一次 LimitedCredit 操作。

➢ **记录文件操作：读记录，写记录和清空记录文件**

int WriteRecord(HANDLE icdev, unsigned char cid, unsigned char mode, unsigned char _FileNo, long _Offset, long _Length, unsigned char *_Data);

功能：卡片当前目录下指定记录文件（线性记录文件或循环记录文件）写数据到一条记录中。

参数：icdev： 通讯设备标识符

cid： 通道号
mode： 数据传输方式，0：明文，1：MAC 码校验，2：DES/3DES 加密
_FileNo： 文件标识符，0x00 到 0x07。
_Offset： 单个记录内的偏移地址
_Length： 写入的数据长度，必须大于 0 小于或等于记录大小－offset
_Data： 写入的数据

返 回：成功则返回 0

例：int st=0;
unsigned char fileno=1;
unsigned char data[15];
memset(data, 0x11, 10);//写入 10 个 0x11
st = WriteRecord(icdev, 0, 0, fileno, 0, 10, data);

注意：在执行该操作之前需要认证写权限或可读/可写权限指定的密钥。
WriteRecord 指令在线性记录文件的末尾添加一条记录，如果文件已满则写入失败。对于循环记录文件在文件的末尾添加一条记录，如果文件已满则擦除并覆盖最老的记录。
如果执行 WriteRecord 之后未执行 CommitTransaction 指令，则下一个 WriteRecord 指令将写上一个 WriteRecord 已经建立的记录。只有在执行了 CommitTransaction 指令后新的 WriteRecord 指令才会建一条新的记录。AbortTransaction 指令可以取消所有修改。

int ReadRecords(HANDLE icdev,unsigned char cid, unsigned char mode, unsigned char _FileNo, unsigned long _Offset, unsigned long _Length, unsigned char *_Data, unsigned long *datalen);

功能：读取卡片当前目录下指定记录文件一条或多条记录信息。

参数：icdev： 通讯设备标识符
cid： 通道号
mode： 数据传输方式，0：明文，1：MAC 码校验，2：DES/3DES 加密
_FileNo： 文件标识符，0x00 到 0x07。
_Offset： 基于最新记录的偏移量，0 到（当前记录数-1）
_Length： 记录数，0 到（当前记录数－_Offset），0 表示读取从最老的记录到由 _Offset 给定的最新记录的所有记录
Data： 读出的数据，长度为记录大小*记录数，如果_Length≠0 返回的是从（当前记录数－_Offset－_Length）开始的_Length 条记录。
_datalen： 返回的数据的长度

返 回：成功则返回 0

例：int st=0;
unsigned char fileno=1;
unsigned char data[50];//假设记录大小为 15 字节
st = ReadRecords(icdev, 0, 0, fileno, 0, 3, data);

注意：在执行该操作之前需要认证读权限或可读/可写权限指定的密钥。
记录文件总是按照记录建立的年代顺序排列，第 0 号记录为最老的记录，最后一

条记录为最新的记录。

int ClearRecordFile(HANDLE icdev, unsigned char cid, unsigned char _FileNo);
功能：将卡片当前目录下指定记录文件清空。
参数：icdev： 通讯设备标识符
cid： 通道号
_FileNo： 文件标识符，0x00 到 0x07。
返 回：成功则返回 0
例：int st=0;
unsigned char fileno=1;
st = ClearRecordFile(icdev, 0, fileno);

注意：在执行该操作之前需要认证可读/可写权限指定的密钥。
在执行 ClearRecordFile 指令之后执行 CommitTransaction 指令之前，写记录将总是失败的，读记录将返回旧的有效的记录；执行 CommitTransaction 指令之后，可以写新的记录，由于没有记录，所以读记录将失败。
执行 AbortTransation 取消清空操作。

➢ **提交/取消当前目录下的备份数据文件、值文件和记录文件前面所有的写访问操作**

int CommitTransaction(HANDLE icdev, unsigned char cid);
功能：提交当前目录下的备份数据文件、值文件和记录文件前面所有的写访问操作，使之前的修改有效。
参数：icdev： 通讯设备标识符
cid： 通道号
返 回：成功则返回 0
例：int st=0;
st = CommitTransaction(icdev, 0);

注意：CommitTransaction 指令使所有基于支持备份机制的文件的写访问有效，支持备份机制的文件类型有备份数据文件，值文件，线性记录文件和循环记录文件。

int AbortTransaction(HANDLE icdev, unsigned char cid);
功能：取消当前目录下的备份数据文件、值文件和记录文件前面所有的写访问操作，使之前的修改无效。
参数：icdev： 通讯设备标识符
cid： 通道号
返 回：成功则返回 0
例：int st=0;
st = AbortTransaction(icdev, 0);

注意：AbortTransaction 指令使所有基于支持备份机制的文件的写访问无效，支持备份机制的文件类型有备份数据文件，值文件，线性记录文件和循环记录文件。