RF 系列非接触式 IC 卡读写器

API 接口函数介绍

目 录

1 简介

1.1 本手册使用范围

本手册描述了非接触式 IC 卡读写器的使用及应用程序接口函数（API），所有 API 函数均可工作于 Windows 98、 Windows 2000、 Windows NT、 Windows XP、 Unix 和 Linux 等操作系统上。

1.2 术语表和缩略语

CRC: 循环码校验
PCD: 近耦合设备
PICC: 近耦合集成电路卡
RWD: 读/写设备
AFI: 应用领域识别号
RFID: 射频识别号
VICC: 近距离集成电路卡
UID: 唯一识别号
DSFID: 数据保存格式识别号
RFU: 保留

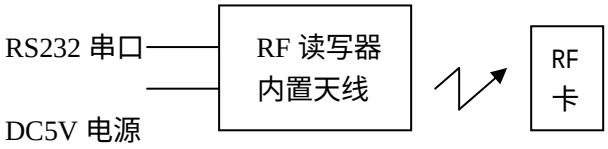
1.3 概述

- 手册简介
- 读写器概述
- API 函数指南
- 通用函数
- 设备操作函数
- Mifare Standard 1K 卡片操作函数
- Mifare UltraLight 卡片操作函数
- 附录（非接触卡特性）

2 读写器概述

RF 非接触式 IC 卡读写器通过 RS232串行接口能实现同 PC 机的连接。随机提供的接口函数库可满足用户二次开发的需要；其完善、可靠的接口函数，支持访问射频卡的全部功能。目前该设备已广泛地应用于门禁、考勤及高速公路、油站、停车场、公交等收费系统中。

2.1 设备接口



RS232 串行接口用于与上位 PC 联机通讯；

2.2 读写器装箱清单

包装盒内配有：读写器，通讯线，5V 电源，驱动软盘。

2.3 程序安装

安装步骤：

a. 将通讯线一端接到读写器上，另一端接至计算机串口上；
b. 接通读写器电源；
c. 打开计算机，进入 WINDOWS 98 或 WINDOWS2000/ME/XP；
d. 将驱动软盘插入驱动器 A ；
e. 执行安装程序；

注：安装结束后，在 Program files 下创建一个 rfreader 的子目录，所有软件均在此目录下。

2.4 软件

RF 读写器软件包括：演示程序、函数库和应用范例

a. 演示程序
提供 Windows 版演示程序: Demo RF.exe。

b. 函数库
C 语言接口函数库
WINDOWS32 位动态库

c. 应用范例
EXAMPLES 目录下提供各种开发平台的应用范例，包括 VB、DELPHI、C 等。

2.5 技术指标

- 操作距离：35mm（Mifare 标准卡读写距离）
- 支持卡型：支持 ISO1443-TypeA 标准，支持 Mifare 标准卡（1K、4K、ML）、Mifare Utral Light 卡（358bit/512bit）、复旦筹码卡 FM005 和华虹卡 SHC1102；
- 数据在卡和读写器之间传输时可进行数据加密和双向验证。（此项功能需要卡片的支持）
- 防冲突，可同时读取多张射频卡（此项功能需要卡片的支持）

- 功能操作：读、写、初始化值、加值、减值、读值和装载密码等（此项功能需要卡片的支持）
- 控制蜂鸣器鸣响功能
- 数码管显示：有 8 位数码管显示
- 通讯接口：RS232 和 RS485 接口
- 波特率：9600~115200bit/s ，自动侦测
- 工作频率 13.56MHZ
- 以 106kbit/s 速率高速访问射频卡
- 工作电源：DC 5V±5%
- 最大功耗：200mW
- 环境温度：0°C~50°C
- 相对湿度：30%~95%
- 重量： 约 200 克
- 提供丰富的二次开发平台和应用范例

3 API 函数指南

3.1 驱动程序主要目录和文件

驱动程序安装后 rfreader 下的目录和文件：

```
Install.log                              安装程序日志
Unwise.exe                               卸载程序
Demo\ DemoRF.exe    WINDOWS 下演示软件
    \ mwrf32.dll        WINDOWS 32 位动态库
Driver\Foxpro.dos\          FOXPRO FOR DOS 驱动程序
Driver\Linux\               Linux 驱动程序
Driver\Unix\                Unix 驱动程序
Driver\Windows\             Windows 驱动程序
Examples\                   VC，VB，DELPHI,Linux,Pb5,Unix,Vfp 等各种平台的范例
```

3.2 函数使用说明

函数调用应遵循如下规则：

（1） 程序开始，首先要调用 rf_init( ) 函数初始化串口。

（2） 用 rf_load_key()将卡中某一扇区密码输入到读写器中。

（3） 调用 rf_card()函数（相当连续调用 rf_request()、anticoll()、select()三个函数），成功可返回卡的序列号。

（4） 用 rf_anthetication() 函数验证设备密码和卡中密码，一次只能验证一个扇区。

（5） 对已验证过的扇区可进行读、写、初始化值、加值、减值等功能操作。对其它扇区的读、写操作必须重复上述（4）过程。

（6） 由于高级函数集成了若干低级函数，所以调用前可不必运行（3)、(4）过程。

（7） 对某张卡操作完成后，应用 rf_halt()函数中止对该卡的操作。

（8） 程序正常退出或因错误退出之前，要用 rf_exit()函数关闭串口；否则，因为串口被占用，再次执行初始化串口时将出错。

（9） **ML 卡**有专用的三个函数 rf_init_ml()、rf_decrement_ml()和 rf_readval_ml；减值以后，再

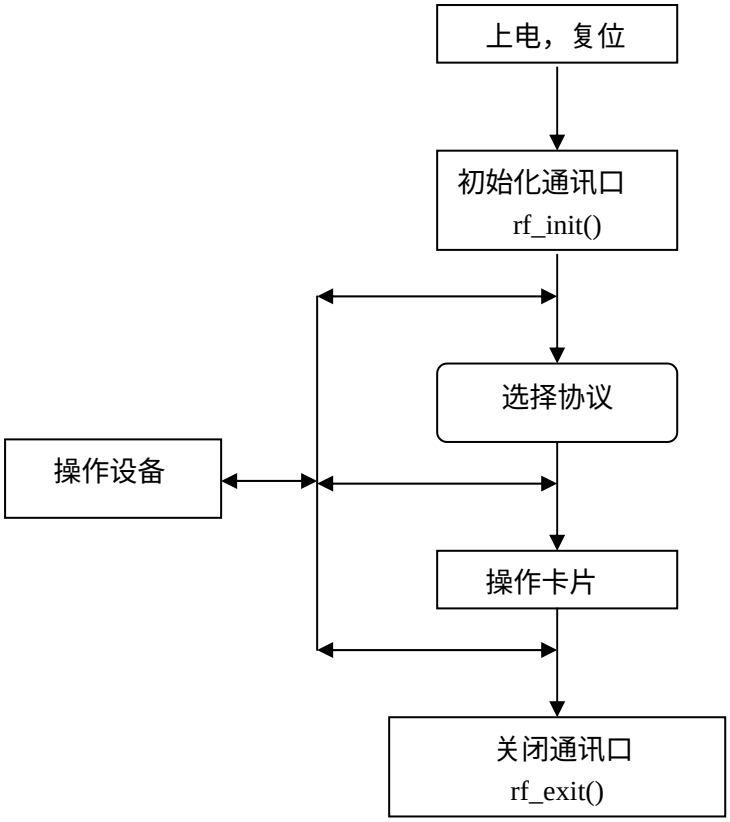
次对卡操作（包括读值）时须重新寻卡。

（10） 有关调用各种函数库的具体方法，请参考:\Examples\目录下的使用范例。

4 库函数简介

本手册主要描述了 RF 系列读写器的 API 函数，包括通用函数、设备操作函数和卡片操作函数。对于不同的操作系统，函数的参数和返回值的数据类型是不同的，并且有些函数是没有的。

Flow Diagram of the Operations on RF series



4.1 通用函数简介

通用函数用来实现打开/关闭串口、加密/解密以及16进制字符串和 ASCII 字符串间的相互转换等。

Windows 函数:

```
int   usb_init();
int   rf_init(int port,long baud);
int   rf_exit(int icdev);
int   rf_decrypt(char *key,unsigned char *ptrSource,unsigned int msglen, char *ptrDest);
int   rf_encrypt(char *key,unsigned char *ptrSource, unsigned int msgLen,unsigned char
          *ptrDest);
int   hex_a(unsigned char *hex,char *a,unsigned char length);
int   a_hex(char *a,unsigned char *hex,unsigned char len);
```

Unix 函数:

```
int rf_init(char *FileName,long baud);
short rf_exit(int icdev);
short rf_encrypt( char *key,char *ptrSource, unsigned short msgLen, char *ptrDest);
short rf_decrypt( char *key,char *ptrSource, unsigned short msgLen, char *ptrDest);
```

Linux 函数:

```
int    rf_init(char *filename,unsigned long baud);
int rf_exit(int icdev);
int rf_encrypt( char *key,char *ptrSource, unsigned long msgLen, char *ptrDest);
int rf_decrypt( char *key,char *ptrSource, unsigned long msgLen, char *ptrDest);
```

4.2 设备操作函数

设备操作函数可以复位读写器、控制蜂鸣器、EEPROM 存储器、获取软件版本号、获取硬件版本号及产品系列号等。

Windows 函数:

控制蜂鸣器：

```
int    rf_beep(HANDLE icdev,unsigned short _Msec);
```

获取硬件版本号:

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

获取产品系列号：

```
int    rf_srd_snr(HANDLE icdev,__int16 lenth,unsigned char *rec_buffer);
```

获取软件版本号：

```
int    lib_ver(unsigned char *str_ver);
```

LED 显示：

```
int    rf_setbright(HANDLE icdev,unsigned char bright);
int    rf_ctl_mode(HANDLE icdev,unsigned char mode);
int    rf_disp_mode(HANDLE icdev,unsigned char mode);
int    rf_disp8(HANDLE icdev,__int16 disp_len,unsigned char* disp_str);
int    rf_disp(HANDLE icdev,unsigned char pt_mode,unsigned short digit);
int    rf_gettime(HANDLE icdev,unsigned char *time);
int    rf_settime(HANDLE icdev,unsigned char *time);
int    rf_gettimehex(HANDLE icdev,char *time);
int    rf_settimehex(HANDLE icdev,char *time);
```

向读写器的 EEPROM 存储器读/写数据：

```
int    rf_srd_eeprom(HANDLE icdev,__int16 offset,__int16 lenth,unsigned char
                *rec_buffer);
int    rf_swr_eeprom(HANDLE icdev,__int16 offset,__int16 lenth,unsigned char*
                send_buffer);
```

Unix 函数:

```
short rf_reset(int icdev,unsigned short _Msec);
short rf_beep(int icdev,unsigned short _Msec);
short rf_get_status(int icdev,unsigned char *_Status);
short rf_srd_snr(int icdev,short lenth,unsigned char *rec_buffer);
```

short lib_ver(unsigned char *str_ver);
short rf_setbright(int icdev,unsigned char bright);
short rf_ctl_mode(int icdev,unsigned char mode);
short rf_disp_mode(int icdev,unsigned char mode);
short rf_disp8(int icdev,short disp_len,unsigned char* disp_str);
short rf_disp(int icdev,unsigned char pt_mode,unsigned short digit);
short rf_gettime(int icdev,unsigned char *time);
short rf_gettimehex(int icdev,char *time);
short rf_settime(int icdev,unsigned char *time);
short rf_settimehex(int icdev,char *time);
short rf_srd_eeprom(int icdev,short offset,short lenth,unsigned char *rec_buffer);
short rf_swr_eeprom(int icdev,short offset,short lenth,unsigned char* send_buffer);

Linux 函数:
int rf_reset(int icdev,unsigned int _Msec);
int rf_beep(int icdev,unsigned short _Msec);
int rf_get_status(int icdev,unsigned char *_Status);
int rf_srd_snr(int icdev,int lenth,unsigned char *rec_buffer);
int lib_ver(unsigned char *str_ver);
int rf_setbright(int icdev,unsigned char bright);
int rf_ctl_mode(int icdev,unsigned char mode);
int rf_disp_mode(int icdev,unsigned char mode);
int rf_disp8(int icdev,int disp_len,unsigned char* disp_str);
int rf_disp(int icdev,unsigned char pt_mode,unsigned short digit);
int rf_gettime(int icdev,unsigned char *time);
int rf_gettimehex(int icdev,char *time);
int rf_settime(int icdev,unsigned char *time);
int rf_settimehex(int icdev,char *time);
int rf_srd_eeprom(int icdev,int offset,int lenth,unsigned char *rec_buffer);
int rf_swr_eeprom(int icdev,int offset,int lenth,unsigned char* send_buffer);

4.3 显示控制函数

Windows 函数:
LED 显示：
int rf_setbright(HANDLE icdev,unsigned char bright);
int rf_ctl_mode(HANDLE icdev,unsigned char mode);
int rf_disp_mode(HANDLE icdev,unsigned char mode);
int rf_disp8(HANDLE icdev,__int16 disp_len,unsigned char* disp_str);
int rf_disp(HANDLE icdev,unsigned char pt_mode,unsigned short digit);
int rf_gettime(HANDLE icdev,unsigned char *time);
int rf_settime(HANDLE icdev,unsigned char *time);
int rf_gettimehex(HANDLE icdev,char *time);
int rf_settimehex(HANDLE icdev,char *time);

Unix 函数:

```
short rf_setbright(int icdev,unsigned char bright);
short rf_ctl_mode(int icdev,unsigned char mode);
short rf_disp_mode(int icdev,unsigned char mode);
short rf_disp8(int icdev,short disp_len,unsigned char* disp_str);
short rf_disp(int icdev,unsigned char pt_mode,unsigned short digit);
short rf_gettime(int icdev,unsigned char *time);
short rf_gettimehex(int icdev,char *time);
short rf_settime(int icdev,unsigned char *time);
short rf_settimehex(int icdev,char *time);
```

Linux 函数:

```
int rf_setbright(int icdev,unsigned char bright);
int rf_ctl_mode(int icdev,unsigned char mode);
int rf_disp_mode(int icdev,unsigned char mode);
int rf_disp8(int icdev,int disp_len,unsigned char* disp_str);
int rf_disp(int icdev,unsigned char pt_mode,unsigned short digit);
int rf_gettime(int icdev,unsigned char *time);
int rf_gettimehex(int icdev,char *time);
int rf_settime(int icdev,unsigned char *time);
int rf_settimehex(int icdev,char *time);
```

4.4 复位 RF（射频）模块

复位射频模块函数将给射频模块断电几毫秒。射频模块复位后，所有在天线区域的卡片都回到上电复位状态。

int rf_reset (HANDLE icdev,unsigned __int16 _Msec);

功能: 将 RF（射频）模块的能量释放几毫秒

参数:

icdev: rf_init()返回的设备描述符

_Msec: 复位时间 (0~ 500ms)

返回: =0: 成功

<>0: 出错

例: st=rf_reset (icdev,60);

4.5 卡片操作

卡片的应用程序接口（API）函数是根据卡片的标准来分类的：

4.5.1 Mifare 标准非接触卡操作函数

Windows 函数:

装载密码**:**

int rf_load_key (int icdev,unsigned char _Mode,unsigned char _SecNr,unsigned char *_NKey);

int rf_load_key_hex (int icdev,unsigned char _Mode,unsigned char _SecNr,char *_NKey);

低级和高级函数都可对 Mifare 卡进行同一操作。每一条高级函数都集成了一系列低级函数。这样用户使用起来会更方便。但是，如果对卡片进行多扇区或多块操作，速度将会变慢，因为在高级函数中许多低级函数的执行是重复的。在这种情况下，我们建议用户调用低级函数。

低级函数:

int rf_request(HANDLE icdev,unsigned char _Mode,unsigned __int16 *TagType);

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

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

int rf_authentication(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr);

int rf_authentication_2(HANDLE icdev,unsigned char _Mode,unsigned char KeyNr,unsigned char Adr);

int rf_read(HANDLE icdev,unsigned char _Adr,unsigned char *_Data);

int rf_read_hex(HANDLE icdev,unsigned char _Adr, char *_Data);

int rf_write(HANDLE icdev,unsigned char _Adr,unsigned char *_Data);

int rf_write_hex(HANDLE icdev,unsigned char _Adr,char *_Data);

int rf_increment(HANDLE icdev,unsigned char _Adr,unsigned long _Value);

int rf_decrement(HANDLE icdev,unsigned char _Adr,unsigned long _Value);

int rf_restore(HANDLE icdev,unsigned char _Adr);

int rf_transfer(HANDLE icdev,unsigned char _Adr);

int rf_initval(HANDLE icdev,unsigned char _Adr,unsigned long _Value);

int rf_readval(HANDLE icdev,unsigned char _Adr,unsigned long *_Value);

int rf_decrement_transfer(HANDLE icdev,unsigned char Adr, unsigned long _Value);

int rf_halt(HANDLE icdev);

高级函数:

int rf_card(HANDLE icdev,unsigned char _Mode,unsigned long *_Snr);

int rf_changeb3(HANDLE icdev,unsigned char _SecNr,unsigned char *_KeyA,unsigned char _B0,unsigned char _B1,unsigned char _B2,unsigned char _B3,unsigned char _Bk,unsigned char *_KeyB);

int rf_check_write(HANDLE icdev,unsigned long Snr,unsigned char authmode,unsigned char Adr,unsigned char * _data);

int rf_check_writehex(HANDLE icdev,unsigned long Snr,unsigned char authmode,unsigned char Adr, char * _data);

int rf_HL_authentication(HANDLE icdev,unsigned char reqmode,unsigned long snr,unsigned char authmode,unsigned char secnr);

int rf_HL_decrement(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long _Snr,unsigned long *_NValue,unsigned long *_NSnr);

int rf_HL_increment(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long _Snr,unsigned long *_NValue,unsigned long *_NSnr);

int rf_HL_write(HANDLE icdev,unsigned char _Mode,unsigned char _Adr,unsigned long *_Snr,unsigned char *_Data);

int rf_HL_writehex(HANDLE icdev,unsigned char _Mode,unsigned char _Adr,unsigned long *_Snr, char *_Data);

int rf_HL_read(HANDLE icdev,unsigned char _Mode,unsigned char _Adr,unsigned long _Snr,unsigned char *_Data,unsigned long *_NSnr);

int rf_HL_readhex(HANDLE icdev,unsigned char _Mode,unsigned char _Adr,unsigned long _Snr, char *_Data,unsigned long *_NSnr);

int rf_HL_initval(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long *_Snr);

Unix 函数:

short rf_request(int icdev,unsigned char _Mode,unsigned short *TagType);

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

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

short rf_authentication(int icdev,unsigned char _Mode,unsigned char _SecNr);

short rf_authentication_2(int icdev,unsigned char _Mode,unsigned char KeyNr,unsigned char Adr);

short rf_read(int icdev,unsigned char _Adr,unsigned char *_Data);

short rf_read_hex(int icdev,unsigned char _Adr,char *_Data);

short rf_write(int icdev,unsigned char _Adr,unsigned char *_Data);

short rf_write_hex(int icdev,unsigned char _Adr,char *_Data);

short rf_initval(int icdev,unsigned char _Adr,unsigned long _Value);

short rf_increment(int icdev,unsigned char _Adr,unsigned long _Value);

short rf_decrement(int icdev,unsigned char _Adr,unsigned long _Value);

short rf_readval(int icdev,unsigned char _Adr,unsigned long *_Value);

short rf_decrement_transfer(int icdev,unsigned char Adr,unsigned long _Value);

short rf_transfer(int icdev,unsigned char _Adr);

short rf_restore(int icdev,unsigned char _Adr);

short rf_halt(int icdev);

short rf_card(int icdev,unsigned char _Mode,unsigned long *_Snr);

short rf_check_write(int icdev,unsigned long Snr,unsigned char authmode,unsigned char Adr,unsigned char * _data);

short rf_check_writehex(int icdev,unsigned long Snr,unsigned char authmode,unsigned char Adr,unsigned char * _data);

short rf_changeb3(int icdev,unsigned char _SecNr,unsigned char *_KeyA,unsigned char _B0,unsigned char _B1,unsigned char _B2,unsigned char _B3,unsigned char _Bk,unsigned char *_KeyB);

short rf_HL_authentication(int icdev,unsigned char reqmode,unsigned long snr,unsigned char authmode,unsigned char secnr);

short rf_HL_decrement(int icdev,unsigned char _Mode,unsigned char _SecNr,
unsigned long _Value,unsigned long _Snr,unsigned
long *_NValue, unsigned long *_NSnr);

short rf_HL_increment(int icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long _Snr,unsigned long *_NValue, unsigned long *_NSnr);

short rf_HL_write(int icdev,unsigned char _Mode,unsigned char _Adr,unsigned long *_Snr,unsigned char *_Data);

short rf_HL_writehex(int icdev,unsigned char _Mode,unsigned char _Adr,unsigned long *_Snr,unsigned char *_Data);

short rf_HL_read(int icdev,unsigned char _Mode,unsigned char _Adr,unsigned long _Snr,unsigned

char *_Data,unsigned long *_NSnr);
short rf_HL_readhex(int icdev,unsigned char _Mode,unsigned char _Adr,unsigned long _Snr,unsigned char *_Data,unsigned long *_NSnr);
short rf_HL_initval(int icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long *_Snr);
short rf_get_snr(int icdev,unsigned char *_Snr);

Linux 函数:

int rf_request(int icdev,unsigned char _Mode,unsigned int *TagType);
int rf_anticoll(int icdev,unsigned char _Bcnt,unsigned long *_Snr);
int rf_select(int icdev,unsigned long _Snr,unsigned char *_Size);
int rf_authentication(int icdev,unsigned char _Mode,unsigned char _SecNr);
int rf_authentication_2(int icdev,unsigned char _Mode,unsigned char KeyNr,
unsigned char Adr);
int rf_read(int icdev,unsigned char _Adr,unsigned char *_Data);
int rf_read_hex(int icdev,unsigned char _Adr,char *_Data);
int rf_write(int icdev,unsigned char _Adr,unsigned char *_Data);
int rf_write_hex(int icdev,unsigned char _Adr,char *_Data);
int rf_initval(int icdev,unsigned char _Adr,unsigned long _Value);
int rf_increment(int icdev,unsigned char _Adr,unsigned long _Value);
int rf_decrement(int icdev,unsigned char _Adr,unsigned long _Value);
int rf_readval(int icdev,unsigned char _Adr,unsigned long *_Value);
int rf_transfer(int icdev,unsigned char _Adr);
int rf_restore(int icdev,unsigned char _Adr);
int rf_decrement_transfer(int icdev,unsigned char Adr,unsigned long _Value);
int rf_halt(int icdev);

int rf_card(int icdev,unsigned char _Mode,unsigned long *_Snr);
int rf_check_write(int icdev,unsigned long Snr,unsigned char authmode,unsigned char Adr,unsigned char * _data);
int rf_check_writehex(int icdev,unsigned long Snr,unsigned char authmode,unsigned char Adr,unsigned char * _data);
int rf_changeb3(int icdev,unsigned char _SecNr,unsigned char *_KeyA,unsigned
char _B0,unsigned char _B1,unsigned char _B2,unsigned char _B3,
unsigned char _Bk,unsigned char *_KeyB);
int rf_HL_authentication(int icdev,unsigned char reqmode,unsigned long snr,unsigned char authmode,unsigned char secnr);
int rf_HL_decrement(int icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long _Snr,unsigned long *_NValue,unsigned long *_NSnr);
int rf_HL_increment(int icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long _Snr,unsigned long *_NValue,unsigned long *_NSnr);
int rf_HL_write(int icdev,unsigned char _Mode,unsigned char _Adr,unsigned long *_Snr,unsigned char *_Data);
int rf_HL_writehex(int icdev,unsigned char _Mode,unsigned char _Adr,unsigned long *_Snr,unsigned char *_Data);

int rf_HL_read(int icdev,unsigned char _Mode,unsigned char _Adr,unsigned long _Snr,unsigned char *_Data,unsigned long *_NSnr);

int rf_HL_readhex(int icdev,unsigned char _Mode,unsigned char _Adr,unsigned long _Snr,unsigned char *_Data,unsigned long *_NSnr);

int rf_HL_initval(int icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long *_Snr);

int rf_get_snr(int icdev,unsigned char *_Snr);

5 通用函数

下面将详细描述通用函数。

1） int usb_init();

功 能：初始化 USB 通讯

参 数：无

返 回：成功则返回设备描述符(≥0)

例：
```
HANDLE icdev;
icdev=usb_init();
```

2） int rf_init(int port,long baud);

功 能: 用选定的与 PC 机通讯的串口和波特率初始化读写器，这是操作读写器的第一步，这样可以获得通讯口 ID 号供以后使用。

参 数: port: 通讯口号(0～3)

Baud: baudrate 通讯波特率(9600～115200)

返 回: ≥0: 成功则返回设备描述符(≥0)

<0: 失败

例:
```
int icdev;
icdev=rf_init(1,115200);// 波特率:115200,端口:com2
```

3) int rf_exit(int icdev);

功 能: 关闭串口并保存 PC 机上的设置。

参 数:

icdev: rf_init()返回的设备描述符

返 回: = 0: 成功

<>0: 失败

例:
```
int st;
st=rf_exit(icdev);
```

4） int rf_encrypt(char *key,unsigned char *ptrSource, unsigned int msgLen, unsigned char *ptrDest);

功 能: DES 算法加密。

参 数:

Key: 密钥，长度为 8个字节

ptrsource: 准备加密的明文,长度必须是8的倍数

msglen: 明文的长度，必须是8的倍数

ptrdest: 密文

返 回: = 0: 成功

<>0: 失败

例: //用 “12345678”加密 “abcdefghabcdefgh”
```
st=rf_encrypt(“1234567”,”abcdefghabcdefgh”,16,ptrdest);
```

5) int rf_decrypt(char *key,unsigned char *ptrSource,unsigned int msglen, char

***ptrDest);**

功 能: DES 算法解密。

参 数:

key: 密钥（必须和加密时的相同），长度为 8个字节

ptrsource: 密文

msglen: 密文的长度，必须是8的倍数

ptrdest: 明文

返 回: =0: 成功

<>0: 失败

例:

//用 “12345678”来解密 “abcdefghabcdefgh”

st=rf_decrypt(“1234567”,”abcdefghabcdefgh”,16,ptrdest);

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

功 能: 将 16 进制数转换为 ASCII 字符。

参 数:

hex: 16 进制数

a: 输出的 ASCII 字符

length: 16 进制数的长度

返 回: =0 成功

<>0 失败

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

功 能: 将 ASCII 字符转换为 16 进制数。

参 数:

a : ASCII 字符

hex: 输出的 16 进制数

length: ASCII 字符的长度

返 回: =0 成功

<>0 失败

6 设备操作函数

1) int rf_reset(HANDLE icdev,unsigned __int16 _Msec);

功 能: 射频头复位(射频头掉电几毫秒)。

参 数:

icdev: rf_init()返回的设备描述符

_Msec: 复位时间，0～500毫秒有效

返 回: =0: 成功

<>0: 失败

例: st=rf_reset(icdev,60);

2) int rf_beep(HANDLE icdev,unsigned short _Msec);

功 能: 蜂鸣几毫秒。

参 数:

icdev: rf_init()返回的设备描述符
_Msec: 蜂鸣时间，单位：毫秒
返 回: = 0: 成功
<>0: 失败
例: st=rf_beep(icdev,10); //鸣叫10毫秒

3) int rf_get_status(HANDLE icdev,unsigned char *_Status);
功 能: 获取读写器的版本号。
参 数:
icdev: rf_init()返回的设备描述符
_Status: 返回读写器版本信息，长度为18字节
返 回: =0: 成功
<>0: 失败
例:
```
int st;
unsigned char status[19];
st=rf_get_status(icdev,status);
```

4) int rf_srd_snr(HANDLE icdev,__int16 lenth,unsigned char *rec_buffer);
功 能: 获取读写器的产品序列号。
参 数:
icdev: rf_init()返回的设备描述符
length: 产品序列号的长度为16字节
receive_buffer: 返回的产品序列号
返 回: =0: 成功
<>0: 失败
例:
```
unsigned char receive_buffer[17];
st=rf_srd_snr(icdev,16,receive_buffer);
```

5) int lib_ver(unsigned char *str_ver);
功 能: 获取 API 函数库版本号。
参 数:
strver: 返回 API 函数库版本号，长度为18个字节
返 回: =0: 成功
<>0: 失败
例：
```
unsigned char str_ver[19];
st=lib_ver(str_ver);
```

6) int rf_srd_eeprom(HANDLE icdev,__int16 offset,__int16 length, unsigned char *rec_buffer);
功 能: 读取 eeprom 的内容。
参 数:
icdev: rf_init()返回的设备描述符
offset: 位移地址 (0-383)
length: 数据长度(1-384)
recv_buffer: 接收数据的缓冲区

返 回: = 0: 成功
<>0: 失败
例: unsigned char Send_buffer[385];
st=rf_srd_eeprom(icdev,0,384,send_buffer);

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

功 能: 向 eeprom 中写入数据。
参 数:
icdev: rf_init()返回的设备描述符
offset: 位移地址(0-383)
length: 数据长度(1-384)
send_buffer: 将写入 eeprom 中的数据
返 回: = 0: 成功
<>0: 失败
例: unsigned char Send_buffer[384];
st=rf_srd_eeprom(icdev,0,384,send_buffer);

7 显示控制操作函数

1) int rf_setbright(HANDLE icdev,unsigned char bright);

功 能: 设置数码管 LED 的亮度
参 数:
icdev: rf_init()返回的设备描述符
bright: 亮度等级 0-15
返 回: =0: 成功
<>0: 失败
例: int bright=7;
st=rf_setbright(icdev,bright);

2) int rf_ctl_mode(HANDLE icdev,unsigned char mode);

功 能: 设置数码管 LED 的显示控制模式
参 数:
icdev: rf_init()返回的设备描述符
mode: 0 由 PC 机控制显示
1 由读写器控制显示(显示时间或日期)
返 回: =0: 成功
<>0: 失败
例:
int mode=0; // 由 PC 机控制显示
st=rf_ctl_mode(icdev,mode);

3) int rf_disp_mode(HANDLE icdev,unsigned char mode);

功 能: 设置数码管 LED 显示时间还是日期（取决于读写器的时钟），需要将 **rf_ctl_mode**

中显示控制模式设为“ 1 ”。

参 数:

icdev: rf_init()返回的设备描述符

mode: 0 显示日期

1 显示时间

返 回: =0: 成功

<>0: 失败

例:

```
int mode=1; //display time
st=rf_disp_mode(icdev,mode);
```

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

功 能: 在数码管 LED 上显示数字内容，需要将 **rf_ctl_mode** 中显示控制模式设为“ 0 ”。

参 数:

icdev: rf_init()返回的设备描述符

disp_len: 显示内容的长度, 必须是8个字节

disp_str: 需要显示的内容

显示的格式取决于将要显示的字符串，如果字节的最高位设为1，该字节后的小数点将会显示。如果字节的最高位不设为1，该字节后的小数点将不会显示。

数码管 LED 的每一位由一个16进制数控制。

16进制数							
B7	B6	B5	B4	B3	B2	B1	B0
1:小数点闪烁	数字 如果是 0x7f, 将不显示。						

返 回: =0: 成功

<>0: 失败

例：

```
//LED 显示:1.234567
unsigned char disp_str[]={0x7f,0x81,0x02,0x03,0x04,0x05,0x06,0x07};
st=rf_disp8(icdev,8,disp_str);
```

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

功 能: 设置数码管 LED 由 PC 机控制来显示数字。

参 数:

icdev: rf_init()返回的设备描述符

pt_mode: 显示小数点的模式

0 不显示小数点

1 显示第一个小数点

2 显示第二个小数点

3 显示第三个小数点

4 显示第四个小数点

digit: 将要显示的数字, 范围： 0 - 9999

返 回: =0: 成功
<>0: 失败

例:
```
int st;
st=rf_disp(icdev,0,1234);        /*显示 1234*/
st=rf_disp(icdev,1,1234);        /*显示 1234.*/
st=rf_disp(icdev,2,1234);        /*显示 123.4*/
```

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

功 能: 读取读写器的日期、星期和时间。

参 数:

icdev: rf_init()返回的设备描述符

receive_data: 接收数据, 长度大于7个字节

receive_data[0]: 年
receive_data[1]: 星期
receive_data[2]: 月
receive_data[3]: 日
receive_data[4]: 时
receive_data[5]: 分
receive_data[6]: 秒

返 回: =0: 成功
<>0: 失败

例:
```
int st;
unsigned char datetime[8];
st=rf_gettime(icdev,datetime);
//datetime =“ 0x99,0x04,0x05,0x20,0x13,0x30,0x10 ”
//1999, Thursday, May 20, 13:30:10
```

7) int rf_settime(HANDLE icdev,unsigned char *time);

功 能: 设置读写器时钟的日期、星期和时间。

参 数:

icdev: rf_init()返回的设备描述符

time: 日期、星期和时间数据

time[0]: 年
time[1]: 星期
time[2]: 月
time[3]: 日
time[4]: 时
time[5]: 分
time[6]: 秒

返 回: =0: 成功
<>0: 失败

例:
```
//设置日期为： 17/06/99 ，时间为：12:34:56, 星期一
unsigned char data[8];
data[0]=0x99;data[1]=0x1;data[2]=0x6;data[3]=0x17;
data[4]=0x12;data[5]=0x34;data[6]=0x56;
```

st=rf_settime(icdev,data);

8) int rf_gettimehex(HANDLE icdev,char *time);

功 能: 读取读写器时钟的日期、星期和时间 (16进制数)。

参 数:

icdev: rf_init()返回的设备描述符

receive_data: 返回的数据，长度大于14个字节

返 回: =0: 成功

<>0: 失败

例: char data[15];

st=rf_gettimehex(icdev,data);

9) int rf_settimehex(HANDLE icdev,char *time);

功 能: 以16进制数设置读写器时钟的日期、星期和时间。

参 数:

icdev: rf_init()返回的设备描述符

time: 时间和日期的数值

返 回: =0: 成功

<>0: 失败

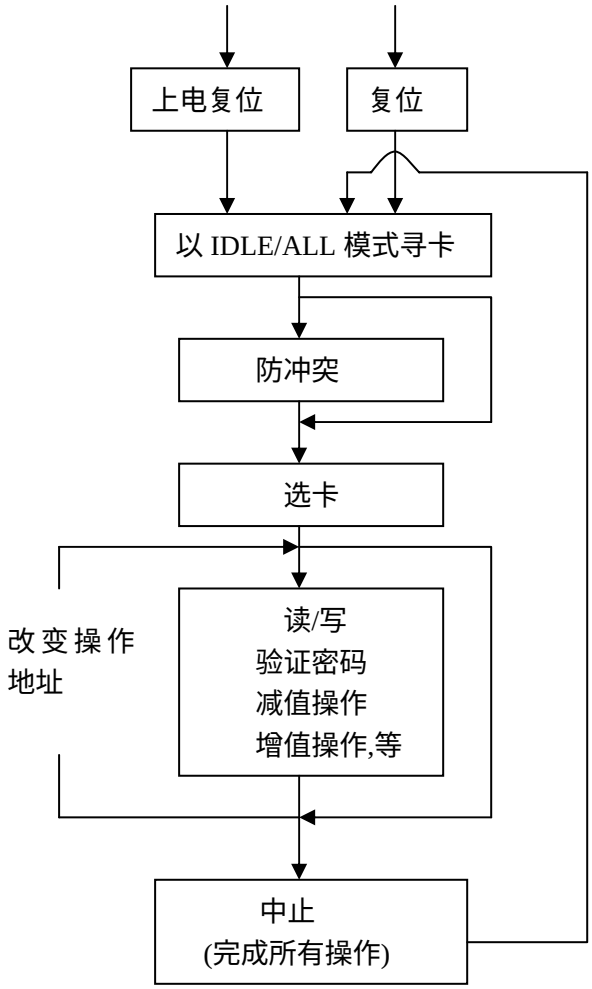
例: //设置日期：17/06/99, 时间：12:34:56, 星期一

char data[14]=“99010617123456”;

st=rf_settimehex(icdev,data);

8. Mifare 标准非接触卡操作函数

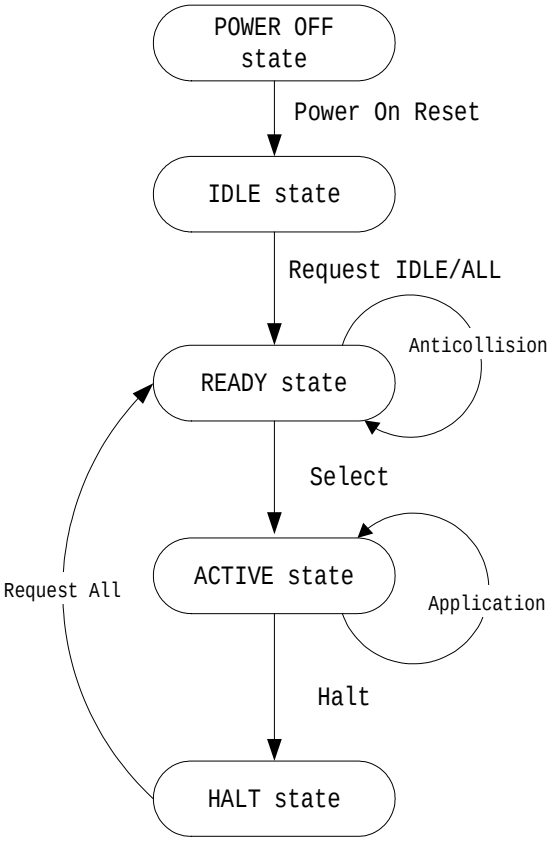
8.1 Mifare 标准非接触卡操作流程图



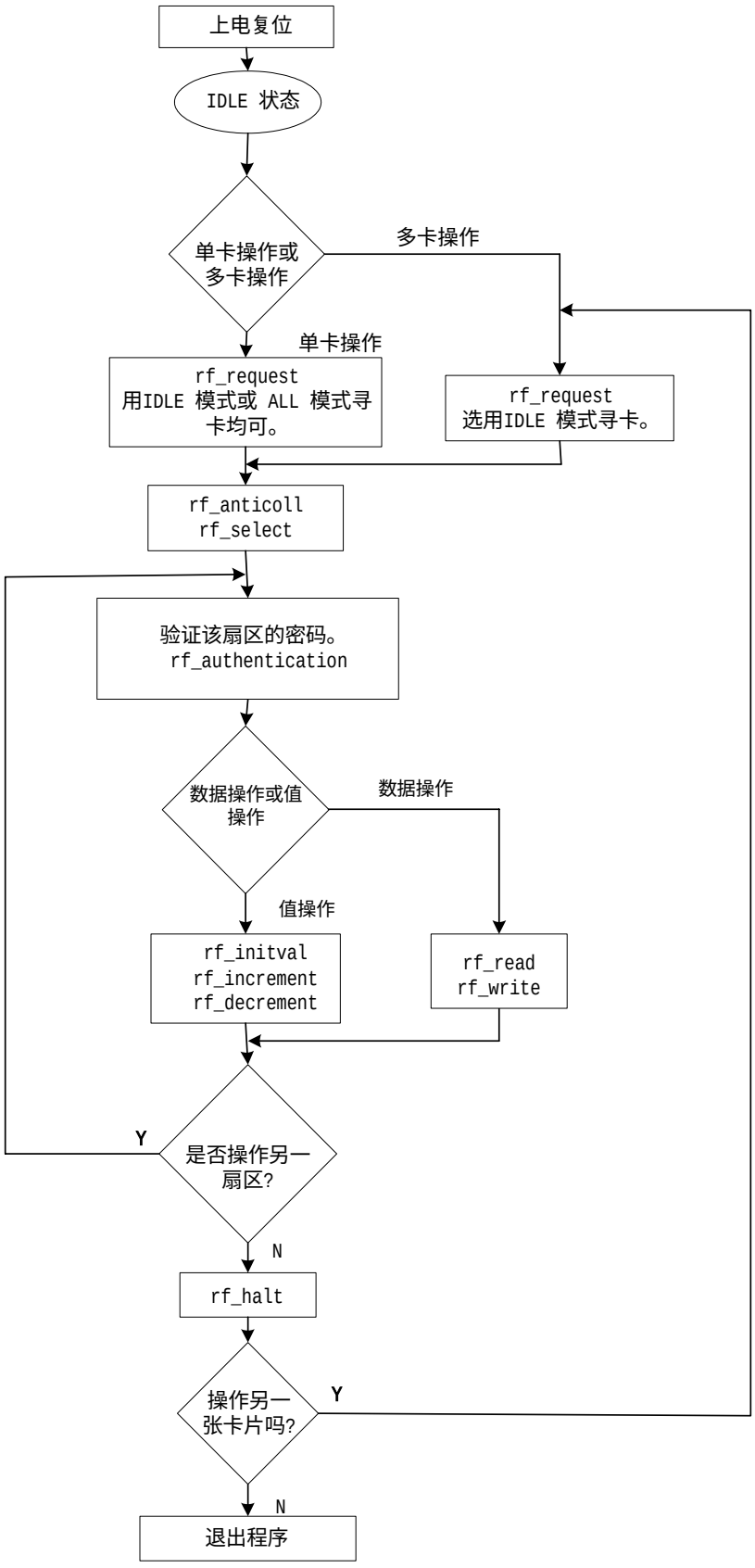
8.2 Mifare Standard 1K 卡片

这里详细介绍了 Mifare Standard 1K 卡片的操作函数，有关卡片的资料详见附录“Mifare Standard 1K”部分。

8.2.1 Mifare Standard 1K 卡片状态图



8.2.2 调用 Mifare Standard 1K 卡片 API 函数流程图



调用 Mifare std 1K 卡片 API函数流程图

8.2.3 操作函数说明

低级和高级函数都可对 Mifare 卡进行同一操作。每一条高级函数都集成了一系列低级函数。这样用户使用起来会更方便。但是，如果对卡片进行多扇区或多块操作，速度将会变慢，因为在高级函数中许多低级函数的执行是重复的。在这种情况下，我们建议用户调用低级函数。

int rf_load_key(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr,unsigned char *_NKey);

功 能: 向读写器装载指定扇区的新密码（不与卡片进行通讯），读写器中有16个扇区的密码（0~15），每个扇区有两个密码(KEY A 和 KEY B)。

参 数:

icdev: rf_init()返回的设备描述符

_Mode: 密码类型

0 — KEY A

4 — KEY B

_SecNr: 须装载密码的扇区号(0～15)

_Nkey: 写入读写器的6字节新密码

返 回: = 0: 成功

<>0: 失败

例： // 装载扇区1的0号 key A：“a0a1a2a3a4a5”

```
unsigned char key[6]= {0xa0,0xa1,0xa2,0xa3,0xa4,0xa5 }
st=rf_load_key(icdev,0,1,key);
```

int rf_load_key_hex(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr,char *_NKey);

功 能: 与 rf_load_key 函数相似。

参 数:

icdev: rf_init()返回的设备描述符

_Mode: 密码类型

0 — KEY A

4 — KEY B

_SecNr: 须装载密码的扇区号(0～15)

_Nkey: 写入读写器的6字节新密码

返 回: =0: 成功

<>0: 失败

例: //装载扇区1的0号 key A：“a0a1a2a3a4a5”

```
char key[ ]= “a0a1a2a3a4a5”;
st=rf_load_key_hex(icdev,0,1,key);
```

8.2.3.1 低级函数

(1) **int rf_request(HANDLE icdev,unsigned char _Mode,unsigned __int16 *TagType);**

功 能: 该函数向卡片发出寻卡命令，开始选择一张新卡片时需要执行该函数。

参 数:

icdev: rf_init()返回的设备描述符

_Mode: 寻卡模式:

0 IDLE mode, 只有处在 IDLE 状态的卡片才响应读写器的命令。

1 ALL mode, 处在 IDLE 状态和 HALT 状态的卡片都将响应读写器的命令。

Tagtype: 返回卡片类型 (Mifare std. 1k: 0x0004, UltraLight: 0x0044, FM005: 0x0005, Mifare std. 4k: 0x0002, SHC1122: 0x3300)

返 回: =0: 成功

<>0: 失败

例：

```
unsigned char Mode=0;
unsigned int tagtype;
st=rf_request(icdev,Mode,&tagtype);
```

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

功 能: 激活读写器的防冲突队列。如果有几张 MIFARE 卡片在感应区内，将会选择一张卡片，并返回卡片的序列号供将来调用 **rf_select** 函数时使用。

参 数:

icdev: rf_init()返回的设备描述符

_Bcnt: 预选卡片使用的位, 标准调用时为 bcnt=0.

_Snr: 返回的卡片序列号

返 回: = 0: 成功

<>0: 失败

例：

```
int st;
unsigned long snr;
st=rf_anticoll(icdev,0,&snr);
```

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

功 能: 用指定的序列号选择卡片，将卡片的容量返回给 PC 机。

参 数:

icdev: rf_init()返回的设备描述符

_Snr: 卡片的序列号

_Size: 卡片容量的地址指针，目前该值不能使用

返 回: = 0: 成功

<>0: 失败

例:

```
int st;
unsigned long snr=239474;
unsigned char size;
st=rf_select(icdev,snr,&size);
```

(4) int rf_authentication(HANDLE icdev,unsigned char _Mode,unsigned char

_SecNr);

功 能: 验证读写器中的密码与需要访问的卡片的同一扇区(0~15)的密码是否一致。如果读写器中选择的密码（可用 **rf_load_key** 函数修改）与卡片的相匹配，密码验证通过，传输的数据将用以下的命令加密。

参 数:

icdev: rf_init()返回的设备描述符

_Mode: 验证密码类型：

0 — 用 KEY A 验证

4 — 用 KEY B 验证

_SecNr: 将要访问的卡片扇区号(0~15)

返 回: = 0: 成功

<>0: 失败

例： int st;

//authentication the 5th sector whit the 0^{th} key A

st=rf_authentication(icdev,0,5);

(5) int rf_authentication_2(HANDLE icdev,unsigned char _Mode, unsigned char KeyNr,unsigned char Adr);

功 能: 验证读写器中的密码与需要访问的卡片的同一扇区(0~15)的密码是否一致。如果读写器中选择的密码（可用 **rf_load_key** 函数修改）与卡片的相匹配，密码验证通过。主要用于验证扇区号大于15的扇区。

参 数:

icdev: rf_init()返回的设备描述符

_Mode: 验证密码类型：

0 — 用 KEY A 验证

4 — 用 KEY B 验证

KeyNr: 读写器中该扇区(0~15)的密码

Adr: 将要访问的卡片块号

返 回: = 0: 成功

<>0: 失败

例: 用读写器中0扇区的 KEY A 验证块2。

int st;

st=rf_authentication_2(icdev,1,0,2);

(6) int rf_read(HANDLE icdev,unsigned char _Adr,unsigned char *_Data);

功 能: 从一张选定并通过密码验证的卡片读取一块共16个字节的数据。

参 数:

icdev: rf_init()返回的设备描述符

_Adr: 读取数据的块号(0~63)

_Data:读取的数据，PC 机上 RAM 的地址空间由调用该函数来分配。

返 回: = 0: 成功

<>0: 失败

例: int st;

unsigned char data[16]；

st=rf_read(icdev,1,data);

(7) int rf_read_hex(HANDLE icdev,unsigned char _Adr, char *_Data);

功 能: 读取16进制数的16 个字节。

参 数:

icdev: rf_init()返回的设备描述符

_Adr: 块地址, 0~63

_Data: 读取的数据

返 回: =0: 成功

<>0: 失败

例:

```
int st;
unsigned char data[32];
//read data from block 1
st=rf_read_hex(icdev,1,data);
```

(8) int rf_write(HANDLE icdev,unsigned char _Adr,unsigned char *_Data);

功 能: 将一块共16字节写入选定并验证通过的卡片中。

参 数:

icdev: rf_init()返回的设备描述符

_Adr: 写入数据的块地址 (1~63)

_Data: 写入数据,长度为16字节

返 回: =0: 成功

<>0: 失败

例:

```
int st;
unsigned char data[16]={0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
                        0x88,0x88,0x77,0x66,0x55,0x44,0x33,0x22,0x11};
st=rf_write(icdev,1,data); // 写入块1
```

(9) int rf_write_hex(HANDLE icdev,unsigned char _Adr,char *_Data);

功 能: 以十六进制写数据, 一次必须写一个块。

参 数:

icdev: rf_init()返回的设备描述符

_Adr: 写入数据的块地址 (1~63)

_Data: 写入数据,长度为32字节

返 回: =0: 成功

<>0: 失败

例:

```
int st;
unsigned char data[32]="a1a2a3a4a5a6a7a8a1a2a3a4a5a6a7a8";
st=rf_write_hex(icdev,1,data); //write block 1
```

(10) int rf_initval(HANDLE icdev,unsigned char _Adr,unsigned long _Value);

功 能: 初始化某一块的值。

参 数:

icdev: rf_init()返回的设备描述符

_Adr: 块地址
_Value: 初始化的目标值
返 回: =0: 成功
<>0: 失败
例:
```
int st;
unsigned long value=1000;
st=rf_initval(icdev,1,value);
```
注：对某一块进行值操作时使用的是特殊的数据结构，所以需要进行初始化，然后才可以进行其它的增值和减值操作。

(11) int rf_increment(HANDLE icdev,unsigned char _Adr,unsigned long _Value);
功 能: 对值操作的块进行增值操作。
参 数:
icdev: rf_init()返回的设备描述符
_Adr: 值操作的块地址
_Value: 增加的值
返 回: = 0: 成功
<>0: 失败
例:
```
int st;
unsigned long value=2;
st=rf_increment(icdev,1,value);
```

(12) int rf_decrement(HANDLE icdev,unsigned char _Adr,unsigned long _Value);
功 能: 对值操作的块进行减值操作。
参 数:
icdev: rf_init()返回的设备描述符
_Adr: 值操作的块地址
_Value: 减少的值
返 回: =0: 成功
<>0: 失败
例:
```
int st;
unsigned long value=2;
st=rf_decrement(icdev,1,value);
```

(13) int rf_readval(HANDLE icdev,unsigned char _Adr,unsigned long *_Value);
功 能: 读出指定值操作块的当前值。
参 数:
icdev: rf_init()返回的设备描述符
_Adr: 值操作的块地址
_Value: 返回读出的值操作块的内容
返 回: = 0: 成功
<>0: 失败
例:
```
int st;
```

```
unsigned long value;
//read the content and putin value
st=rf_readval(icdev,1,&value);
```

(14) int rf_restore(HANDLE icdev,unsigned char _Adr);

功 能: 将某块的数据传入卡的内部寄存器中。

参 数:

icdev: rf_init()返回的设备描述符

_Adr: 卡片上将读出数据的块地址

返 回: =0: 成功

<>0: 失败

例:
```
int st;
st=rf_restore(icdev,1);
```

注：用此函数将某一块内的数值传入卡的内部寄存器，然后用 rf_transfer()函数

将寄存器的数据再传送到另一块中去，即实现了块与块之间的数值传送。

(15) int rf_transfer(HANDLE icdev,unsigned char _Adr);

功 能: 将内部寄存器的数据传送到某一块中。进行此项操作必须验证该扇区的密码，在执行 increment, decrement 或 restore 操作后可直接调用。

参 数:

icdev: rf_init()返回的设备描述符

_Adr: 内部寄存器的内容将存放的地址。.

返 回: =0 : 成功

<>0: 失败

例:
```
int st;
st=rf_transfer(icdev,1);
```

(16) int rf_decrement_transfer(HANDLE icdev,unsigned char Adr, unsigned long _Value);

功 能: 通过传送来减少块的值。

参 数:

icdev: rf_init()返回的设备描述符

_Adr: 块地址

_Value: 减少的值

返 回: =0: 成功

<>0: 失败

例:
```
int st;
unsigned long value=2;
st=rf_decrement_transfer(icdev,1,value);
```

(17) int rf_halt(HANDLE icdev);

功 能:将一张选中的卡片设为“Halt”模式，只有当该卡再次复位或用 ALL 模式调用 request 函数时，读写器才能够再次操作它。

参 数:
icdev: rf_init()返回的设备描述符
返 回: =0: 成功
<>0: 失败
例: st=rf_halt(icdev);
注 ：使用 rf_card()函数时，如果模式选择为 0 则在对卡进行读写操作完毕后，必须执行 rf_halt()，且只能当该卡离开并再次进入操作区域时，读写器才能够再次操作它。

8.2.3.2 高级函数

(1) int rf_card(HANDLE icdev,unsigned char _Mode,unsigned long *_Snr);
功 能: 寻卡并返回卡片的系列号，它可以完成低级函数 rf_request, rf_anticoll 和 rf_select 的功能。
参 数:
icdev: rf_init()返回的设备描述符
_Mode: 寻卡模式
0: IDLE 模式，一次只操作一张卡
1: ALL 模式，一次可操作多张卡
_Snr: 返回卡片的系列号
返 回: =0 : 成功
<>0 : 失败
例:
```
int st;
unsigned char Mode=0; //IDLE mode
unsigned long snr;
st=rf_card(icdev,Mode,&snr);
```

注：rf_card()是三个低级函数的组合:rf_request(),rf_select() 和 rf_anticoll()。

注意：选用 IDLE 模式寻卡时，完成对卡片的操作后调用 rf_halt 函数来停止操作，此后读写器不能找到卡片，除非卡片离开操作区域并再次重新进入。
选用 ALL 模式寻卡时，完成对卡片的操作后调用 rf_halt 函数来停止操作，此后读写器仍能找到该卡片，无须离开操作区域并再次重新进入。

(2) int rf_changeb3(HANDLE icdev,unsigned char _SecNr,unsigned char *_KeyA,unsigned char _B0,unsigned char _B1,unsigned char _B2,unsigned char _B3,unsigned char _Bk,unsigned char *_KeyB);
功 能: 修改 KeyA, 访问条件和 KeyB.
参 数:
icdev: rf_init()返回的设备描述符
_SecNr: 扇区号
_KeyA: key A

_B0: 0块的控制位，低三位（D2D1D0）对应为 C10,C20,C30.
_B1: 1块的控制位，（D2D1D0）对应为 C11,C21,C31
_B2: 2块的控制位，（D2D1D0）对应为 C12,C22,C32
_B3: 3块的控制位，（D2D1D0）对应为 C13,C23,C33
_Bk: 保留参数,设为0.
_KeyB: key B

返 回: =0: 成功
<>0: 失败

例:
```
int st;
unsigned char keya[6]={0xa0,0xa1,0xa2,0xa3,0xa4,0xa5};
unsigned char keyb[6]={0xb0,0xb1,0xb2,0xb3,0xb4,0xb5};
st=rf_changeb3(icdev,keya,0x04,0x04,0x04,0x04,0,keyb);
```

(3) int rf_check_write(HANDLE icdev,unsigned long Snr,unsigned char authmode,unsigned char Adr,unsigned char * _data);

功 能: 检查写入卡片的内容，在执行 rf_write () 函数后调用该函数。

参 数:

icdev: rf_init()返回的设备描述符
Snr: 卡片系列号
Authmode: 密码验证模式
0 用 A 密码验证
1 用 B 密码验证
Adr: 块地址
_data: 检查的内容.

返 回: =0: 成功
<>0 失败

例：
```
unsigned char databuff[]={0x00,0x11,0x22,0x33,0x44,0x55,0x66,
                          0x77,0x88,0x88,0x77,0x66,0x55,0x44,0x33,0x22,0x11};
unsigned char authmode=0;
st=rf_write(icdev,4,databuff); // 写入第4块
// 检查第4块的内容正确与否
st=rf_check_write(icdev,authmode,4,databuff);
```

(4) int rf_check_writehex(HANDLE icdev,unsigned long Snr,unsigned char authmode,unsigned char Adr, char * _data);

功 能: 与 rf_check_write () 函数类似，但使用的是16进制数。

参 数:

icdev: rf_init()返回的设备描述符
Snr: 卡片系列号
Authmode: 密码验证模式
Adr: 块地址
_data: 检查的内容

返 回: = 0: 成功
<>0: 失败

例: unsigned char data[32]=”00112233445566778899aabbccddeeff”;
unsigned char authmode=0;
st=rf_write_hex(icdev,4,data);
st=rf_check_writehex(icdev,authmode,4,data);

(5) int rf_HL_initval(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long *_Snr);

功 能: 高级初始化值（只用于扇区不用于块）
参 数:
icdev: rf_init()返回的设备描述符
Mode: 高级函数有三种模式
0——IDLE 模式 ，一次只操作一张卡
1——ALL 模式 ，一次可操作多张卡
2——选择模式 ，只操作选中的卡片
_SecNr: 扇区号（0～15）
_Value: 初始化的值
_Snr: 卡片系列号（只在模式2，选择模式中使用）
返 回: = 0: 成功
<>0: 失败
函数操作流程: request（ALL 或 IDLE）
anticoll（SEL=0）
select
authentication
write（DATA）
write（BACKUP）
read （DATA）
read （BACKUP）
compare
halt
例: unsigned long snr;
st=rf_HL_initval(icdev,0x0,3,100L,&snr);

(6) int rf_HL_decrement(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long _Snr,unsigned long *_NValue,unsigned long *_NSnr);

功 能: 高级减值操作（用于扇区）
参 数:
icdev: rf_init()返回的设备描述符
_NValue：将要减去的值
其余参数参见 rf_hl_initval () 函数.
返 回: = 0: 成功
<>0: 失败
函数操作流程: request（ALL 或 IDLE）
anticoll（SEL=0）
select

authentication
read（DATA）
read（BACKUP）
compare
decrement（DATA）
transfer（BACKUP）
restore（BACKUP）
transfer（DATA）
halt

例: unsigned long Snr,Nvalue,NSnr;
st=rf_HL_decrement(icdev,0,2,1,Snr,&Nvalue,&NSnr);

(7) int rf_HL_increment(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr,unsigned long _Value,unsigned long _Snr,unsigned long *_NValue,unsigned long *_NSnr);

功 能: 高级增值操作(用于扇区)

参 数:

_Nvalue: rf_init()返回的设备描述符
其余参数参见 rf_hl_initval () 函数.

返 回: =0: 成功
<>0: 失败

函数操作流程: request（ALL 或 IDLE）
anticoll（SEL=0）
select
authentication
read（DATA）
read（BACKUP）
compare
increment（DATA）
transfer（BACKUP）
restore（BACKUP）
transfer（DATA）
halt

例： unsigned char Snr,Nvalue,NSnr;
st=rf_HL_increment(icdev,0,2,1,Snr,&Nvalue,&NSnr);

(8) int rf_HL_write(HANDLE icdev,unsigned char _Mode,unsigned char _Adr,unsigned long *_Snr,unsigned char *_Data);

功 能: 高级写函数，向选定的并通过密码验证的卡片写入1块16个字节。

参 数:

icdev: rf_init()返回的设备描述符
_Mode: 寻卡模式，与 rf_HL_initval () 函数相似
_Adr: 块地址
_Snr: 卡片系列号（仅用于模式2）

_Data: 写入卡片的数据 (长度为16 字节)

返 回: =0: 成功

<>0: 失败

函数操作流程: request（ALL 或 IDLE）

anticoll（SEL=0）

select

authentication

write（_Adr）

read（_Adr）

halt

例： unsigned long Snr;

unsigned char data[16]="f1f2f3f4f5f6f7f8";

st=rf_HL_write(icdev,0,3,&Snr,data);

(9) int rf_HL_writehex(HANDLE icdev,unsigned char _Mode,unsigned char _Adr,unsigned long *_Snr, char *_Data);

功 能: 16进制高级写操作。

返 回: = 0: 成功

<>0: 失败

例: unsigned char data[32]="f1f2f3f4f5f6f7f8f1f2f3f4f5f6f7f8";

unsigned long Snr;

st=rf_HL_writehex(icdev,0,3,&Snr,data);

(10) int rf_HL_read(HANDLE icdev,unsigned char _Mode,unsigned char _Adr,unsigned long _Snr,unsigned char *_Data,unsigned long *_NSnr);

功 能: 高级读函数，从选定的并通过密码验证的卡片读出1块16个字节。

参 数:

icdev: rf_init()返回的设备描述符

_Mode: 寻卡模式，与 rf_HL_initval ()函数相似

_Adr: 块地址

_Snr: 卡片系列号（仅用于模式2）

_Data: 从卡片中读出的数据 (长度为16 字节)

_NSnr: 返回卡片系列号

返 回: = 0: 成功

<>0: 失败

函数操作流程: request（ALL 或 IDLE）

anticoll（SEL=0）

select

authentication

read（_Adr）

read (BACKUP)

compare

halt

例: unsigned long Snr,NSnr;

```
unsigned char data[16];
st=rf_HL_read(icdev,0,3,Snr,data,&NSnr);
```

(11) int rf_HL_readhex(HANDLE icdev,unsigned char _Mode,unsigned char _Adr,unsigned long _Snr, char *_Data,unsigned long *_NSnr);

功 能: 高级16进制读操作

返 回: = 0: 成功

<>0: 失败

例:
```
unsigned char data[32];
unsigned long Snr,NSnr;
st=rf_HL_readhex(icdev,0,3,Snr,data,&NSnr);
```

(12) int rf_HL_authentication(HANDLE icdev,unsigned char reqmode, unsigned long snr,unsigned char authmode,unsigned char secnr);

功 能: 高级验证函数(组合了 rf_card() 和 rf_authentication() 函数)

参 数:

icdev: rf_init()返回的设备描述符

reqmode: 寻卡模式，与 rf_HL_initval ()函数相似

snr: 卡片系列号（仅用于模式2）

authmode: 密码验证模式

0 —用 A 密码验证

4 —用 B 密码验证

secnr: 扇区号（0～15）

返 回: =0: 成功

<>0 失败

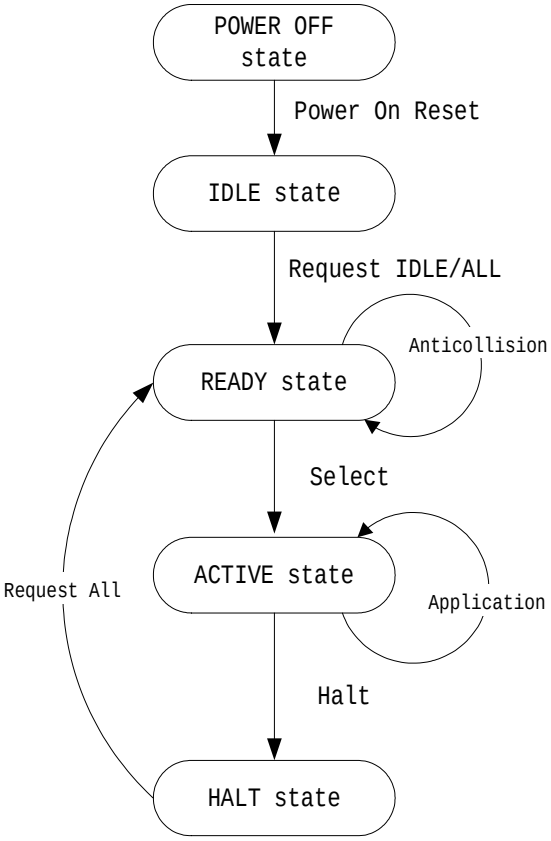
例:
```
//用 IDLE 模式
unsigned long snr;
st=rf_HL_authentication(icdev,0,snr,0,3);
```

在 mifare 系列卡的操作函数中，针对同一个操作有高级函数和低级函数之分。

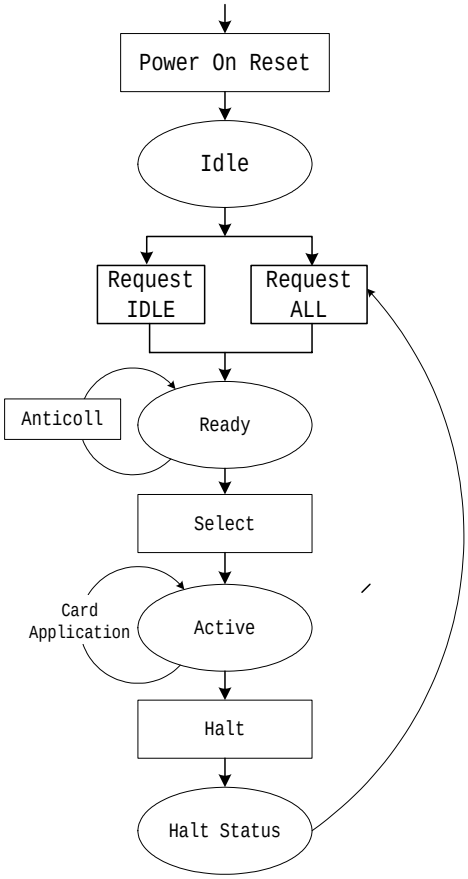
8.3 Mifare Standard 4K

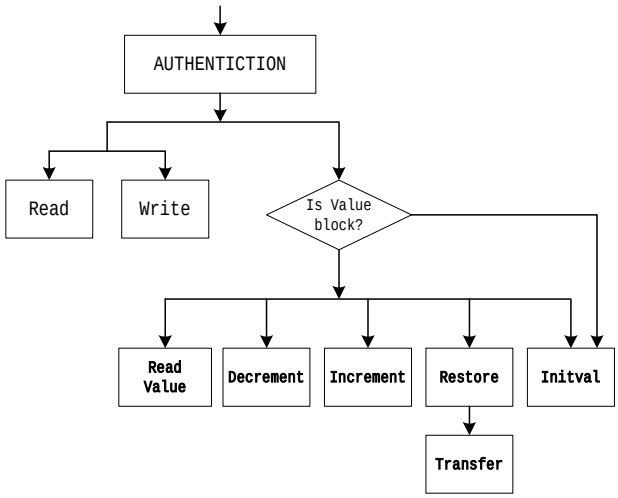
这里只说明了 Mifare Standard 4K 卡的操作函数，有关 Mifare Standard 4K 卡的资料请参考附录。

8.3.1 状态图和指令流程



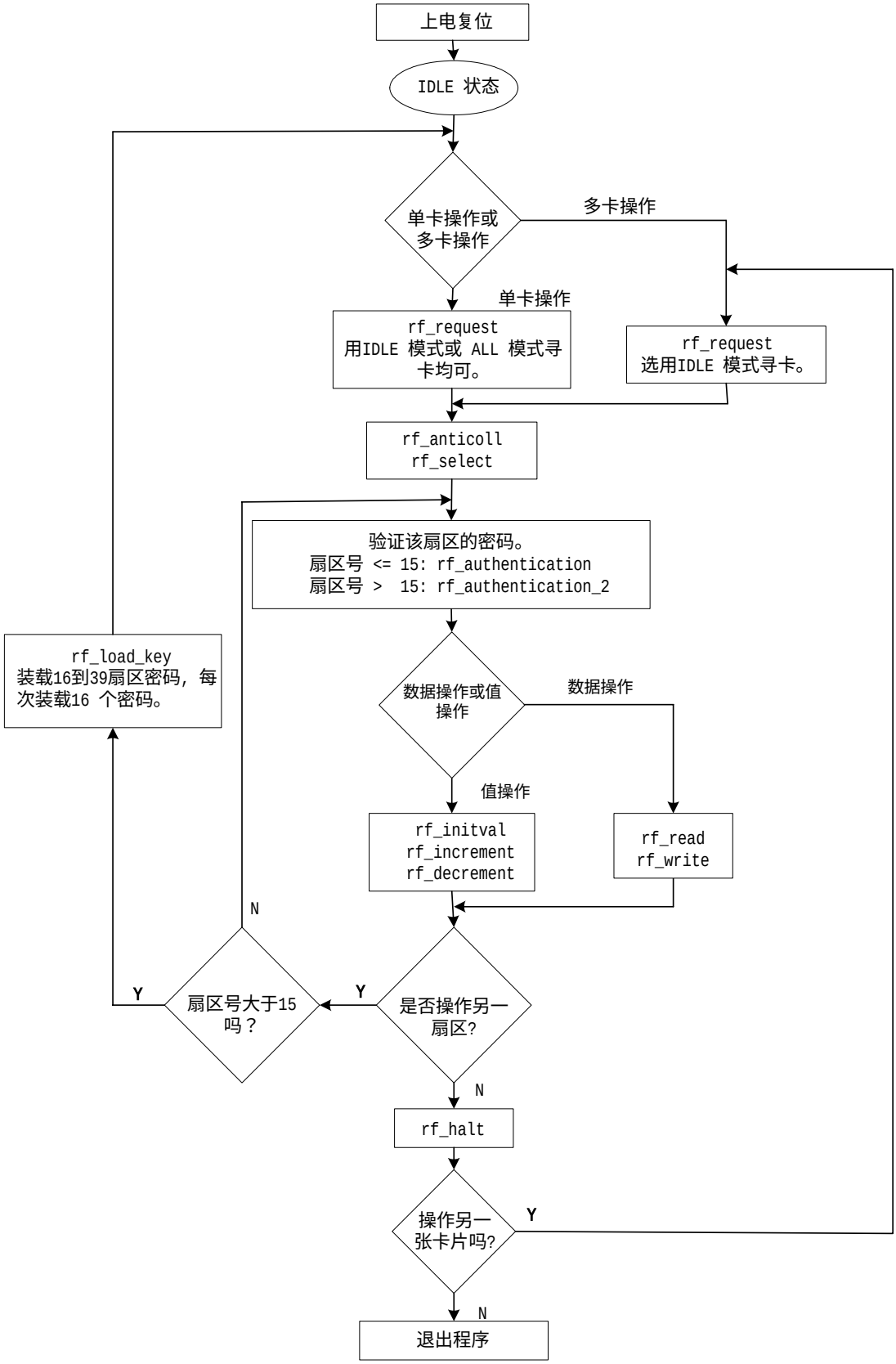
8.3.2 操作流程图





Flow Diagram of Card Application

8.3.3 函数说明:



调用 Mifare std 4K 卡片 API函数流程图

在 mifare 系列卡的操作函数中，针对同一个操作有高级函数和低级函数之分。

8.3.3.1 低级函数

(1) int rf_request(HANDLE icdev,unsigned char _Mode,unsigned __int16 *TagType);

功 能：寻卡请求

参 数：icdev：rf_init()返回的设备描述符

_Mode：寻卡模式

0 IDLE mode, 只有处在 IDLE 状态的卡片才响应读写器的命令。

1 ALL mode, 处在 IDLE 状态和 HALT 状态的卡片都将响应读写器的命令。

Tagtype：卡类型值,(Mifare std. 1k: 0x0004, UltraLight: 0x0044, FM005: 0x0005, Mifare std. 4k: 0x0002, SHC1122: 0x3300)

返回值: = 0: 成功

<>0: 失败

例：
```
#define IDLE 0x00
int st;
unsigned int *tagtype;
st=rf_request(icdev,IDLE,tagtype);
```

注意：关于寻卡模式指令 ：

如果选择 IDLE 模式寻卡对卡进行读写操作，执行 rf_halt()指令中止卡操作后，只有当该卡离开并再次进入操作区时，读写器才能够再次寻到这张卡。

如果选择 ALL 模式寻卡对卡进行操作,执行 rf_halt()命令中止卡操作后,卡可以不必离开操作区.读写器下次也能寻到那张相同的卡.

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

功 能：激活读写器的防冲突队列。如果有几张 MIFARE 卡片在感应区内，将会选择一张卡片,并返回卡片的序列号供将来调用 **rf_select** 函数时使用。

参 数：icdev：rf_init()返回的设备描述符

_Bcnt： 设为 0

_Snr：返回的卡序列号地址

返 回：成功则返回 0

例：
```
int st;
unsigned long snr;
st=rf_anticoll(icdev,0,&snr);
```

注：request 指令之后应立即调用 anticoll，除非卡的序列号已知。

(3) 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，返回卡的容量
参 数：icdev：rf_init()返回的设备描述符
_Snr：卡序列号
_Size：指向返回的卡容量的数据
返 回：成功则返回 0

例: int st;
unsigned long snr=239474;
unsigned char size;
st=rf_select(icdev,snr,&size);

(4) int rf_authentication(HANDLE icdev,unsigned char _Mode,unsigned char _SecNr);

功 能：验证某一扇区密码
参 数：icdev：rf_init()返回的设备描述符
_Mode：密码验证模式
0 — 用 A 密码验证
4 — 用 B 密码验证
SecNr：要验证密码的扇区号（0～15）
返 回：成功则返回 0

example：int st;
//authentication the 5 sector whit the 0^{th} key A
st=rf_authentication(icdev,0,5);

注：RF 系列的读写器只能装载 16 个扇区的密码。16-39 扇区必须调用 rf_authentication_2()验证。

(5) int rf_authentication_2(HANDLE icdev,unsigned char _Mode, unsigned char KeyNr,unsigned char Adr);

功 能:用0—15扇区的密码来验证指定的扇区
参 数:
icdev：rf_init()返回的设备描述符
_Mode: 验证模式 ，与 rf_authentication 相同
KeyNr: 密码扇区号 (0~15)
Adr: 要验证的扇区地址 (0~39).
返回值: = 0: 成功
<>0: 失败
例: int st;
//authentication with the 1th key A 。
st=rf_authentication_2(icdev,0,0,0);

(6) int rf_read(HANDLE icdev,unsigned char _Adr,unsigned char *_Data);

功 能：读取卡中数据,一次读一个块的数据，为 16 个字节；
参 数：icdev：rf_init()返回的设备描述符

_Adr：M1 卡——块地址（0～255）；
_Data：读出数据
返 回：成功则返回 0
例：int st;
unsigned char data[17];
st=rf_read(icdev,4,data); //读 M1 卡块 4 的数据

(7) int rf_read_hex(HANDLE icdev,unsigned char _Adr, char *_Data);
功 能: 以十六进制形式读取数据；
参 数:
icdev：rf_init()返回的设备描述符
_Adr： 块地址（0～255）；
_Data：读出数据
返回值: =0: 成功
<>0: 失败
例: int st;
unsigned char data[33];
//读取块1的数据
st=rf_read_hex(icdev,1,data);

(8) int rf_write(HANDLE icdev,unsigned char Adr,unsigned char *Data);
功 能：向卡中写入数据,一次必须写一个块，为 16 个字节；
参 数：
icdev：rf_init()返回的设备描述符
_Adr：M1 卡—块地址（1～255）；
_Data：要写入的数据，长度为 16 字节
返回值: =0: 成功
<>0: 失败
例: int st;
unsigned char data[16]={0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
0x88,0x88,0x77,0x66,0x55,0x44,0x33,0x22,0x11};
st=rf_write(icdev,1,data); //写块1

(9) int rf_write_hex(HANDLE icdev,unsigned char _Adr,char *_Data);
功 能: 用十六形式进写
参 数:
icdev：rf_init()返回的设备描述符
_Adr: 块地址(1~255)
_Data: 要写入的数据 ,长度是 32
返回值: =0: 成功
<>0: 失败
例: int st;
unsigned char data[32]="a1a2a3a4a5a6a7a8a1a2a3a4a5a6a7a8";
st=rf_write_hex(icdev,1,data); //写块1

(10) int rf_initval(HANDLE icdev,unsigned char _Adr,unsigned long

_Value);

功 能：初始化块值

参 数：icdev：rf_init()返回的设备描述符

_Adr：块地址

_Value：初始值

返 回：成功则返回 0

例：
```
int st;
unsigned long value;
value=1000;                    /* 给 value 赋值*/
st=rf_initval(icdev,1,value);   /*将块 1 的值初始化为 1000*/
```

注：在进行值操作时，必须先执行初始化值函数，然后才可以读、减、加的操作

(11) int rf_increment(HANDLE icdev,unsigned char _Adr,unsigned long _Value);

功 能：块加值

参 数：icdev：rf_init()返回的设备描述符

_Adr：块地址

_Value：要增加的值

返 回：成功则返回 0；

例：
```
int st;
unsigned long value;
value=10;
```

(12) int rf_decrement(HANDLE icdev,unsigned char _Adr,unsigned long _Value);

功 能：块减值

参 数：icdev：rf_init()返回的设备描述符

_Adr：块地址

_Value：要减的值

返 回：成功则返回 0

例：
```
int st;
unsigned long value;
value=10;
            st=rf_decrement(icdev,1,value);
```

(13) int rf_readval(HANDLE icdev,unsigned char _Adr,unsigned long *_Value);

功 能：读块值

参 数：icdev：rf_init()返回的设备描述符

_Adr：块地址

_Value：读出值的地址

返 回：成功则返回 0

例：
```
int st;
unsigned long value;
st=rf_readval(icdev,1,&value);    /*读出块 1 的值，放入 value*/
```

(14) int rf_restore(HANDLE icdev,unsigned char _Adr);

功 能：回传函数，将 EEPROM 中的内容传入卡的内部寄存器

参 数：icdev：rf_init()返回的设备描述符
_Adr：要进行回传的块地址
返 回：成功返回 0
例：int st;
st=rf_restore(icdev,1);

注：用此函数将某一块中的数值传入内部寄存器，然后用 rf_transfer()函数将寄存器中数据再传送到另一块中去，实现块与块之间数值传送。该函数只用于值块。

(15) int transfer(HANDLE icdev,unsigned char _Adr);

功 能：传送，将寄存器的内容传送到 EEPROM 中，在 rf_restore（）后执行.
参 数：icdev：rf_init()返回的设备描述符
_Adr：要传送的地址
返 回：成功返回 0
例：rf_restore(icdev,1);
rf_transfer(icdev,2);
上两行实现将块 1 的内容传送到块 2。

（16）int rf_halt(HANDLE icdev);

功 能：中止对该卡操作
参 数：icdev：rf_init()返回的设备描述符
返 回：成功则返回 0
例：st=rf_halt(icdev);
说明：执行该命令后如果是 ALL 寻卡模式则必须重新寻卡才能够对该卡操作，如果是 IDLE 模式则必须把卡移开感应区再进来才能寻得这张卡。

8.3.3.2 高级函数

（1）int rf_card(HANDLE icdev,unsigned char _Mode,unsigned long *_Snr);

功 能：寻卡并返回卡片的系列号，它可以完成低级函数 rf_request, rf_anticoll 和 rf_select 的功能。
参 数：icdev：rf_init()返回的设备描述符
_Mode：寻卡模式
0: IDLE 模式
1: ALL 模式
_Snr：返回的卡序列号
返 回：成功则返回 0
例：int st;
unsigned long snr;

st=rf_card(icdev,0,&snr);IDLE 模式寻卡

注意：rf_card() 由三个低级函数组成 :rf_request(),rf_select() and rf_anticoll()。

如果选择 IDLE 模式寻卡对卡进行读写操作，执行 rf_halt()指令中止卡操作后，只有当该卡离开并再次进入操作区时，读写器才能够再次寻到这张卡。

如果选择 ALL 模式寻卡对卡进行操作,执行 rf_halt()命令中止卡操作后,卡可不必离开操作区.读写器下次也能寻到那张相同的卡

(2) int rf_changeb3(HANDLE icdev,unsigned char _SecNr,unsigned char *_KeyA,unsigned char _B0,unsigned char _B1,unsigned char _B2,unsigned char _B3,unsigned char _Bk,unsigned char *_KeyB);

功 能：修改块 3 的数据

参 数：icdev：rf_init()返回的设备描述符

_SecNr：扇区号

_KeyA：密码 A

_B0：块 0 控制字，低 3 位（D2D1D0）对应 C10、C20、C30

_B1：块 1 控制字，低 3 位（D2D1D0）对应 C11、C21、C31

_B2：块 2 控制字，低 3 位（D2D1D0）对应 C12、C22、C32

_B3：块 3 控制字，低 3 位（D2D1D0）对应 C13、C23、C33

_Bk：保留参数，取值为 0

_KeyB：密码 B

返 回：成功则返回 0

例：

```
int st;
 int st;
unsigned char keya[6]={0xa0,0xa1,0xa2,0xa3,0xa4,0xa5};
unsigned char keyb[6]={0xb0,0xb1,0xb2,0xb3,0xb4,0xb5};
st=rf_changeb3(icdev,keya,0x04,0x04,0x04,0x04,0,keyb);
```

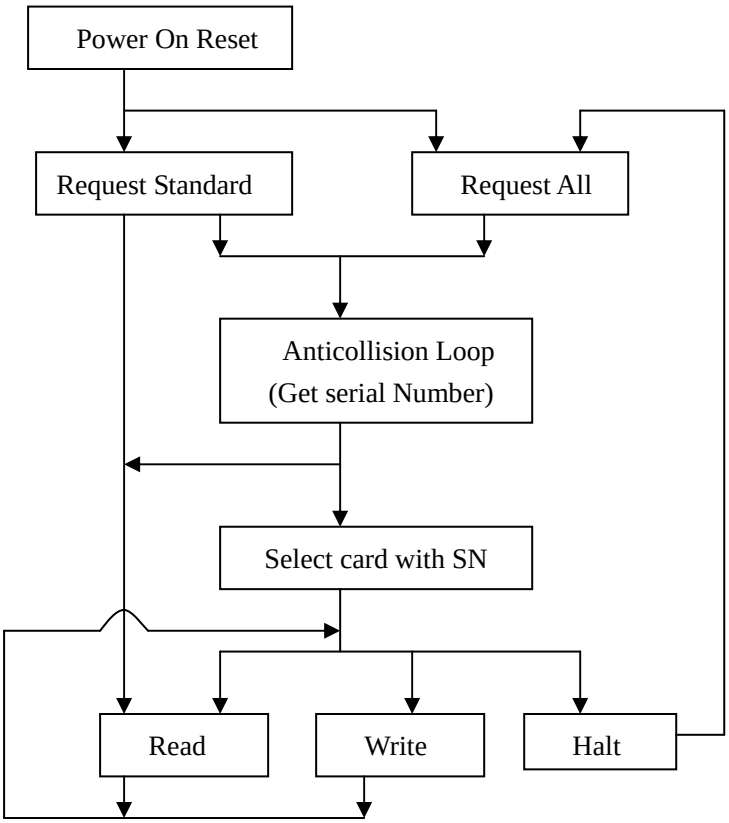
8.3.4 非 Mifare 4K 卡片操作

国产 4K 卡有些个标准 4K 卡不是完全兼容的，例如复旦的 4K 卡是包含 64 个相等的扇区，每个扇区都是 4 个块，其操作跟上面相同，只是扇区地址不同。

8.4 Mifare UltraLight

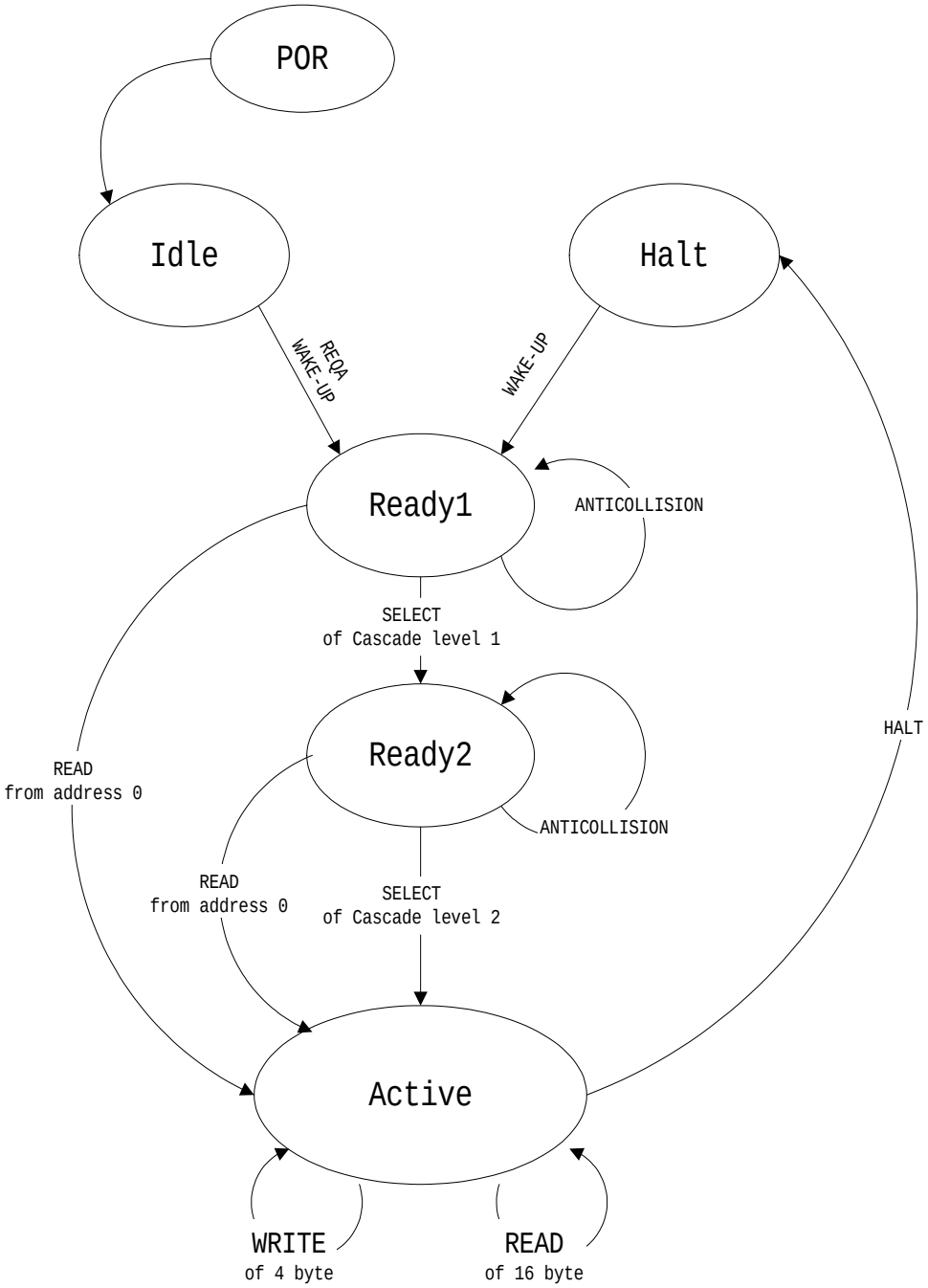
这里只说明了 Mifare UltraLight 卡的操作函数，有关 Mifare UltraLight 卡的资料请参考附录。

8.4.1 操作流程图



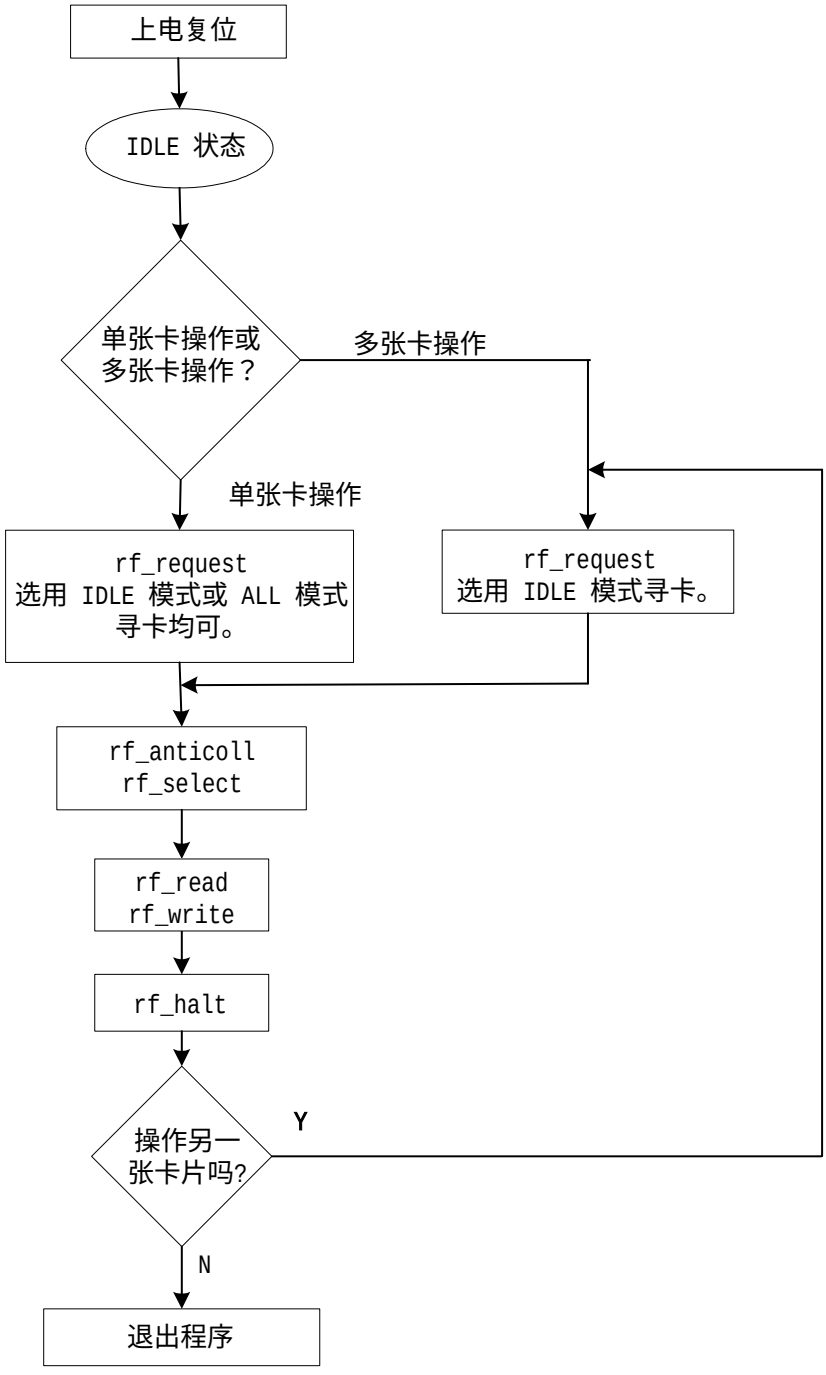
SN: serial number

8.4.2 Mifare UltraLight 状态图



Note:Not shown in this diagram: In each state the command interpreter returns to the Idle state if an unexpected command is received.If the IC has already been in the Halt state vefore it returns to the Halt state in such a case.

8.4.3 函数说明



调用 Mifare UltraLight 卡片 API函数流程图

1) int rf_request(HANDLE icdev,unsigned char _Mode,unsigned __int16 *TagType);

功 能：寻卡请求

参 数：icdev：rf_init()返回的设备描述符

_Mode：寻卡模式

0 IDLE mode, 只有处在 IDLE 状态的卡片才响应读写器的命令。

1 ALL mode, 处在 IDLE 状态和 HALT 状态的卡片都将响应读写器的命令。

Tagtype：卡类型值,(Mifare std. 1k: 0x0004, UltraLight: 0x0044, FM005: 0x0005, Mifare std. 4k: 0x0002, SHC1122: 0x3300)

返回值: = 0: 成功

<>0: 失败

例：
```
#define IDLE 0x00
int st;
unsigned int *tagtype;
st=rf_request(icdev,IDLE,tagtype);
```

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

功 能：激活读写器的防冲突队列。如果有几张 MIFARE 卡片在感应区内，将会选择一张卡片，并返回卡片的序列号供将来调用 **rf_select** 函数时使用。

参 数：icdev：rf_init()返回的设备描述符

_Bcnt： 设为 0

_Snr：返回的卡序列号地址

返 回：成功则返回 0

例：
```
int st;
unsigned long snr;
st=rf_anticoll(icdev,0,&snr);
```

注：request 指令之后应立即调用 anticoll，除非卡的序列号已知。

3) 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

参 数：icdev：rf_init()返回的设备描述符

_Snr：卡序列号

_Size：指向返回的卡容量的数据

返 回：成功则返回 0

例：
```
int st;
unsigned long snr=239474;
unsigned char size;
st=rf_select(icdev,snr,&size);
```

4) int rf_get_snr(HANDLE icdev,unsigned char *_Snr);

功 能：取 UltraLight 卡片序列号，此种卡的序列号长度为 7 字节

参 数：icdev：rf_init()返回的设备描述符

_Snr： 返回的卡片系列号

_Snr[0]　卡片序列号第 0 字节
. .
. .
_Snr[6]　卡片序列号第 6 字节

返回值: =0: 成功
<>0: 失败

例: int st;
unsigned char _Snr[8];
st = rf_get_snr(icdev, _Snr);

5) int rf_read(HANDLE icdev,unsigned char _Adr,unsigned char *_Data);

功 能：读取卡中数据,一次读一页的数据 ；

参 数:
icdev：rf_init()返回的设备描述符
_Adr: 页地址(0～15)
_Data: 读取的数据

返回值: = 0: 成功
<>0: 失败

例: int st;
unsigned char data[17]；
st=rf_read(icdev,1,data);

6) int rf_read_hex(HANDLE icdev,unsigned char _Adr, char *_Data);

功 能:与 rf_read（）相同，读出的数据以十六进制形式表示

参库数:
icdev：rf_init()返回的设备描述符
_Adr: 页地址(0～15)
_Data: 读取的数据，以十六进制形式表示

返回值: =0: 成功
<>0: 失败

例: int st;
unsigned char data[32];
//读页1的数据
st=rf_read_hex(icdev,1,data);

7) int rf_write(HANDLE icdev,unsigned char _Adr,unsigned char *_Data);

功 能：向卡中写入一个长度为 16 字节的数据，但是只有低 4 位的字节写入到指定的地址空间中，建议 4-15 字节设为“0”。

参 数：icdev：rf_init()返回的设备描述符
_Adr： 页地址 ；
_Data：要写入的数据，长度为 16 字节，一页的长度为 4 字节，剩下的 12 个字节初始化为“0”；

返回值: =0: 成功
<>0: 失败

例: int st;

```
unsigned char data[17];
memet(data, 0, 17);
memcpy(data, “\x11\x22\x33\x44”);
st=rf_write(icdev,2,data);              //写第二页
```

8) int rf_write_hex(HANDLE icdev,unsigned char _Adr,char *_Data);

功 能: 用十六进制的形式写

参 数:

icdev：rf_init()返回的设备描述符

_Adr： 页地址 ；

_Data: 写入的数据,长度为32字节. 剩下的24 个字节初始化为 “ 0 ”；

返回值: =0: 成功

<>0: 失败

例:

```
int st;
unsigned char data[33];
memset(data, 0, 33);
memcpy(data,"a1a2a3a4”,8);
st=rf_write_hex(icdev,1,data);              //写第1页
```

9) int rf_halt(HANDLE icdev);

功 能：中止对该卡操作，执行这个指令后，在重新复位之前，不能再对卡进行通讯，除非 rf_request（）的寻卡模式为 ALL。

参 数：icdev：rf_init()返回的设备描述符

返回值: =0: 成功

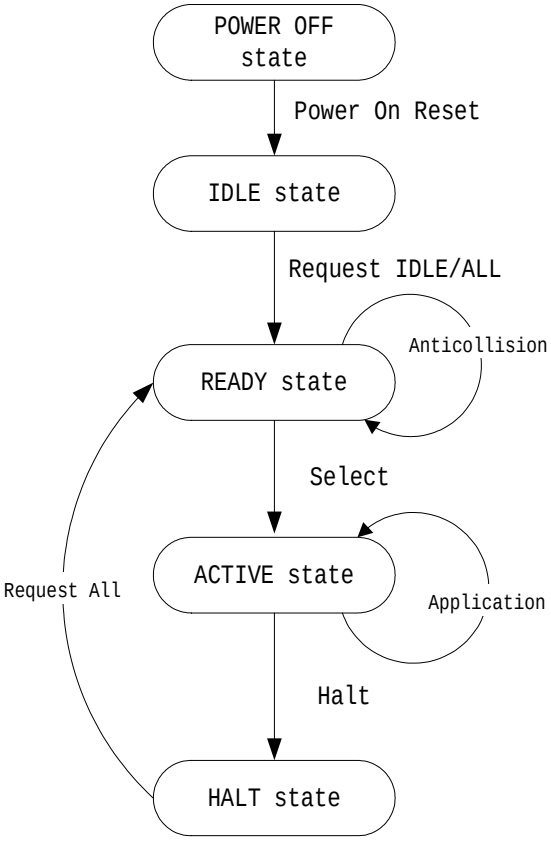
<>0: 失败

例: st=rf_halt(icdev);

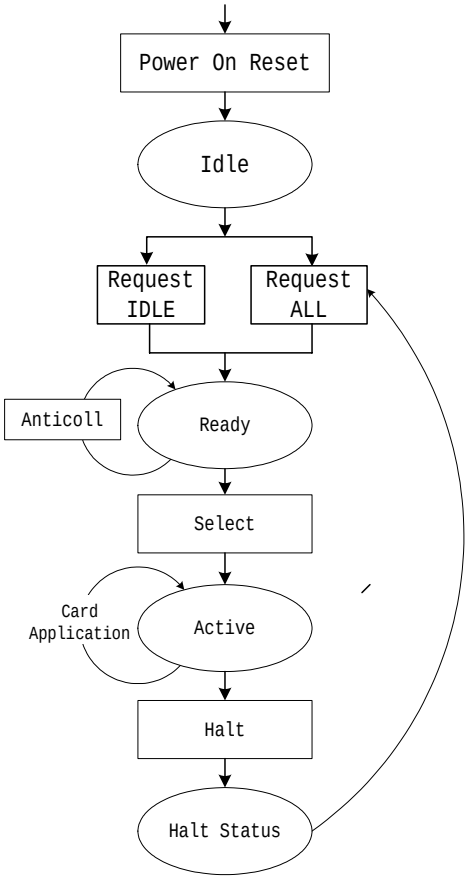
8.5 MifarePRO 卡

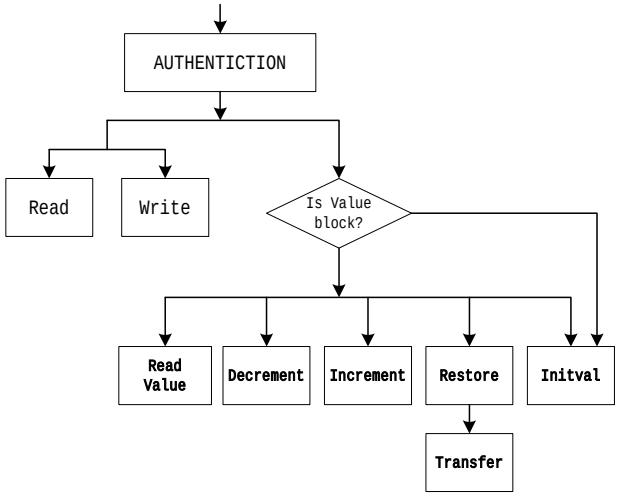
MifarePRO 卡是双界面的 CPU 卡，接触界面遵循 ISO7816 标准，支持 T=0 和 T=1 协议；非接触界面遵循 ISO14443-2 标准，支持 T=CL 协议。有关接触界面的操作请参考 CPU 卡操作函数。

8.5.1 状态图和指令流程



8.5.2 操作流程图

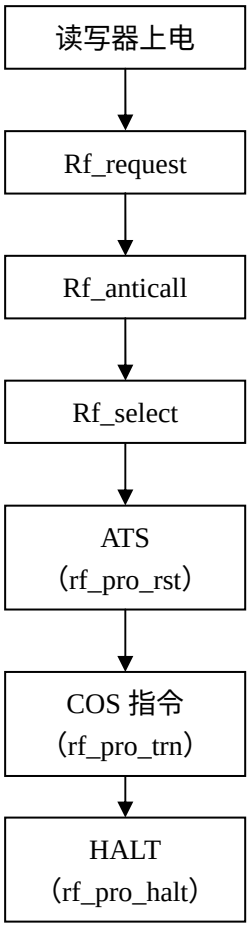




Flow Diagram of Card Application

8.5.3 函数说明

8.5.3.1 函数操作步骤



8.5.3.2 函数说明

通用非接触函数 rf_request、rf_anticall、rf_select 等参见 M1 卡操作。

1) **__int16 __stdcall rf_pro_rst(HANDLE icdev,unsigned char *_Data);**

功 能：非接触 CPU 卡复位。

参 数：icdev：rf_init()返回的设备描述符

_Data：返回的卡复位应答信息

返 回：成功则返回 0

例：
```
int st;
unsigned char data[80];
st = rf_pro_rst(icdev,data);
```

2) **__int16 rf_pro_trn(HANDLE icdev,unsigned char *problock,unsigned char *recv);**

功 能：非接触 CPU 卡 COS 指令操作。

参 数：icdev：rf_init()返回的设备描述符

problock：向卡发送的 COS 指令

recv：卡返回的数据

返 回：成功则返回 0

例：
```
int st;
memset(send,0,256);
memset(recv,0,256);
send[0] = 0;
send[1] = 0;
send[2] = 0;
send[3] = 5;//命令长度
memcpy(&send[4],"\x00\x84\x00\x00\x08",5);//取随机数
st = rf_pro_trn(icdev,send,recv);
```

3) **__int16 __stdcall rf_pro_halt(HANDLE icdev);**

功 能：中止对该卡操作，执行这个指令后，在重新复位之前，不能再对卡进行通讯，除非 rf_request（）的寻卡模式为 ALL。

参 数：icdev：rf_init()返回的设备描述符

返 回：成功则返回 0

例：
```
int st;
st = rf_pro_halt(icdev);
```

9 CPU 卡和 SAM 卡操作函数

9.1 CPU 卡操作函数

1) **__int16 __stdcall rf_cpu_rst(HANDLE icdev, unsigned char baud, unsigned char *cpuack)；**

功 能： CPU 卡复位。

参 数：icdev：rf_init()返回的设备描述符

baud：复位波特率（1：9600，2：19200，3：38400，4：76800，5：153600）

cpuack：返回的卡复位应答信息

返 回：成功则返回 0

例：
```
int st;
 unsigned char rst_baud=1;
 unsigned char data[80];
st=rf_cpu_rst(icdev,rst_baud,data);
```

2) **__int16 __stdcall rf_cpu_trn(HANDLE icdev, unsigned char *cpublock,unsigned char *recv)；**

功 能： CPU 卡指令复位。

参 数：icdev：rf_init()返回的设备描述符

cpublock：发给卡的 COS 指令

recv：返回的卡操作信息

返 回：成功则返回 0

例：
```
int st;
  static unsigned char pcb=0;
  unsigned char send[20],recv[32];
  send[0]=0x00;
  send[1]=pcb;
  send[2]=0x05;
  send[3]=0x00;
  send[4]=0x84;
  send[5]=0x00;
  send[6]=0x00;
  send[7]=0x08;
  send[8]=0;
  for(int i=0;i<8;i++)
        send[8]=send[8]^send[i];
 //取随机数
 st = rf_cpu_trn(icdev,send,recv);
```

9.2 SAM 卡操作函数

1) **__int16 __stdcall rf_sam_rst(HANDLE icdev, unsigned char baud, unsigned char *samack)；**

功 能： SAM 卡复位。
参 数：icdev：rf_init()返回的设备描述符
baud：复位波特率（1：9600，2：19200，3：38400，4：76800，5：153600）
samack：返回的卡复位应答信息
返 回：成功则返回 0
例：int st;
unsigned char rst_baud=1;
unsigned char data[80];
st=rf_sam_rst(icdev,rst_baud,data);

2) __int16 __stdcall rf_sam_trn(HANDLE icdev, unsigned char *samblock,unsigned char *recv)；

功 能： SAM 卡指令复位。
参 数：icdev：rf_init()返回的设备描述符
samblock：发给卡的 COS 指令
recv：返回的卡操作信息
返 回：成功则返回 0
例：int st;

```
static unsigned char pcb=0;
unsigned char send[20],recv[32];
send[0]=0x00;
send[1]=pcb;
send[2]=0x05;
send[3]=0x00;
send[4]=0x84;
send[5]=0x00;
send[6]=0x00;
send[7]=0x08;
send[8]=0;
for(int i=0;i<8;i++)
    send[8]=send[8]^send[i];
//取随机数
st = rf_sam_trn(icdev,send,recv);
```

3) __int16 __stdcall rf_sam_off(HANDLE icdev)；

功 能： SAM 卡下电。
参 数：icdev：rf_init()返回的设备描述符
返 回：成功则返回 0
例：int st;
st=rf_sam_off(icdev);

附录　非接触卡片的特性

1　　MIFARE STANDARD 1K

特性:

- 1K 字节 EEPROM
- 分为16个扇区，每个扇区包括4 块，每块16个字节，以块为存取单位
- 用户可自定义每个存储块的访问条件
- 每张卡有唯一序列号，为32位
- 具有防冲突机制，支持多卡操作
- 非接触传送数据和无源 (卡中无电源)
- 至少10 年数据保存期
- 至少10万次擦写
- 读写距离: 在100mm 内(与天线形状有关)
- 工作频率: 13.56 MHZ
- 通信速率: 106kbit/s
- 典型交易过程: <100 ms(包括备份管理)
- 温度范围: -20℃~50℃

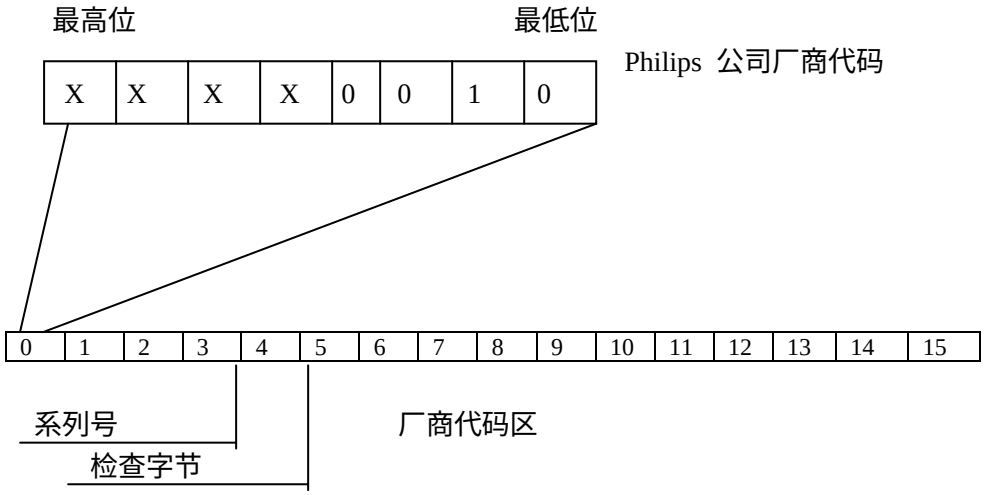
存储结构:

1. 1024*8 位 EEPROM 存储区分为16 个扇区，每扇区分为4块(块0, 块1, 块2, 块3)，按块号编址为0～63共64块。存储区的分布图如下:

扇区号	序号	1 2 3 4 5	6 7 8 9	10 11 12 13 14 15	名称	块号
	0				厂商代码区	0
0	1				数据区	1
	2				数据区	2
	3	A 密码	存取控制	B 密码	扇区 0 控制块区	3
	0				数据区	4
1	1				数据区	5
	2				数据区	6
	3	A 密码	存取控制	B 密码	扇区 1 控制块区	7
		⋮				

	0				数据区	60
15	1				数据区	61
	2				数据区	62
	3	A 密码	存取控制	B 密码	扇区 15 控制块区	63

厂商代码区: 第0扇区的块0（即绝对地址0块）用于存放厂商代码，已经固化，不可更改。



数据区: 所有扇区都有3块（每块16个字节）存储数据。 (扇区0只有两个数据块和一个只读厂商代码块)

值块: 值块可用作电子钱包（有效的命令有：read,write,increment,decrement,restore,transfer）.每个值块的值为4个字节。

2. 扇区控制块(块 3): 每个扇区都有一个扇区控制块包括：
   - 密码 A 和密码 B(可选), 读取时返回 “0”
   - 访问该扇区4块的存取控制

   如果不需要密码 B，块3的最后6个字节可用作数据。

控制属性

各扇区的块 0、块 1、块 2 为**数据块**，用于存贮数据；块 3 为**控制块**，存放密码 A、存取控制、密码 B，其结构如下：

A0A1A2A3A4A5　FF 07 80 69　B0B1B2B3B4B5

密码 A(6 字节)　存取控制(4 字节)　密码 B(6 字节)

每个扇区的密码和存取控制都是独立的，可以根据实际需要设定各自的密码及存取控制。在**存取控制**中每个块都有相应的三个**控制位**,定义如下：

块 0：	C10	C20	C30
块 1：	C11	C21	C31
块 2：	C12	C22	C32
块 3：	C13	C23	C33

三个控制位以正和反两种形式存在于存取控制字节中，决定了该块的访问权限（如进行减值

操作必须验证 KEY A，进行加值操作必须验证 KEY B，等等)。三个控制位在存取控制字节中的位置如下（字节 9 为备用字节，默认值为 0x69）：

	bit 7	6	5	4	3	2	1	0
字节 6	C23_b	C22_b	C21_b	C20_b	C13_b	C12_b	C11_b	C10_b
字节 7	C13	C12	C11	C10	C33_b	C32_b	C31_b	C30_b
字节 8	C33	C32	C31	C30	C23	C22	C21	C20

（注： _b 表示取反）

其中，黑色区控制**块 3**，蓝色区控制**块 2**，绿色区控制**块 1**，红色区控制**块 0**。

数据块（块 0、块 1、块 2）的存取控制如下：

控制位(X=0..2)			**访 问 条 件 （对块 0、1、2）**			
C1X	**C2X**	**C3X**	**Read**	**Write**	**Increment**	**Decrement transfer restore**
0	0	0	KeyA|B	KeyA|B	KeyA|B	KeyA|B
0	1	0	KeyA|B	Never	Never	Never
1	0	0	KeyA|B	KeyB	Never	Never
1	1	0	KeyA|B	KeyB	KeyB	KeyA|B
0	0	1	KeyA|B	Never	Never	KeyA|B
0	1	1	KeyB	KeyB	Never	Never
1	0	1	KeyB	Never	Never	Never
1	1	1	Never	Never	Never	Never

（KeyA|B 表示密码 A 或密码 B，Never 表示任何条件下不能实现）

例如：当块 0 的存取控制位 C10 C20 C30=100 时，验证密码 A 或密码 B 正确后可读；验证密码 B 正确后可写；不能进行加值、减值操作。

控制块（块 3）的存取控制与**数据块**（块 0、1、2）不同，它的存取控制如下：

控制位			**密码 A**		**存取控制**		**密码 B**	
C13	**C23**	**C33**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**
0	0	0	Never	KeyA|B	KeyA|B	Never	KeyA|B	KeyA|B
0	1	0	Never	Never	KeyA|B	Never	KeyA|B	Never
1	0	0	Never	KeyB	KeyA|B	Never	Never	KeyB
1	1	0	Never	Never	KeyA|B	Never	Never	Never
0	0	1	Never	KeyA|B	KeyA|B	KeyA|B	KeyA|B	KeyA|B

0	1	1	Never	KeyB	KeyA|B	KeyB	Never	KeyB
1	0	1	Never	Never	KeyA|B	KeyB	Never	Never
1	1	1	Never	Never	KeyA|B	Never	Never	Never

例如：当块 3 的存取控制位 C13 C23 C33=100 时，表示：

密码 A： 不可读，验证 KEYB 正确后，可写（更改）。

存取控制：验证 KEYA 或 KEYB 正确后，可读不可写。

密码 B： 不可读，验证 KEYB 正确后，可写。

工作原理

卡片的电气部分只由一个天线和 ASIC 组成。

天线：卡片的天线是只有几组绕线的线圈，很适于封装到 IS0 卡片中。

ASIC：卡片的 ASIC 由一个高速(106KB 波特率)的 RF 接口,一个控制单元和一个 8K 位 EEPROM 组成。

读写器向 M1 卡发一组固定频率的电磁波，卡片内有一个 LC 串联谐振电路，其频率与讯写器发射的频率相同，在电磁波的激励下，LC 谐振电路产生共振，从而使电容内有了电荷，在这个电容的另一端，接有一个单向导通的电子泵，将电容内的电荷送到另一个电容内储存，当所积累的电荷达到 2V 时，此电容可做为电源为其它电路提供工作电压，将卡内数据发射出去或接取读写器的数据。

功能介绍：

1. **Request Standard/ALL:** 一张卡上电复位后就可以响应读写器发出的寻卡命令。读写器向天线范围内的所有卡片发出命令，并识别卡片的型号。
2. **Anticollision Loop:**在防冲突循环中将读出卡片的系列号。如果有几张卡片都在读写器的读写范围内，可以通过唯一的系列号区别它们，并选中其中一张卡片，其余没有选中的卡片将进入等待状态，等待下一次寻卡命令。
3. **Select Card:** 用选卡命令读写器选择一张卡片来进行密码验证和有关的存储操作。卡片将对代码为08h 的选卡命令 ATS 做出响应，该命令决定了所选卡片的类型。
4. **3 Pass Authentication:** 读写器选中一张卡片后就指定了后续存取访问的存储空间，并以次响应开始3重密码验证。
5. **HALT:** 调用 rf_halt () 函数来停止对卡片的所有操作，卡片进入 HALT 状态。
6. **Memory Operations:** 密码验证通过后，可进行以下操作：
   - **读块:** 读取一个存储块的内容
   - **写块:** 写入一个存储块的内容.
   - **减值:** 减少一个块的值并保存在内部寄存器内
   - **增值:** 增加一个块的值并保存在内部寄存器内
   - **保存:** 将块的内容写入数据寄存器中
   - **传输:** 将内部寄存器的内容写入某一块中

   注:用 rf_ restore () 函数将某一块的内容传到内部寄存器内，然后用 rf_truansfer() 函数将内部寄存器的内容传到卡片其余块中。通过这种方法，可以将数据从一块传向另一块。

2 MIFARE UlTRALIGHT

特性:

- 容量为512位，分为16页，每页4个字节
- 每页可编程锁定只读功能
- 32位用户可定义的一次性编程区域
- 384位用户读、写区域
- 唯一的7字节序列号
- 非接触传送数据和无源 (卡中无电源)
- 读写距离：在100mm 以内（与天线有关）
- 工作频率：13.56MHZ
- 通信速率：106KB 波特率
- 完整的数据格式: 16 位 CRC, 奇偶校验,位编码,位计数
- 防冲突：同一时间可处理多张卡
- 数据可保留2年
- 可循环改写1000次
- 典型交易过程: <35 ms(包括备份管理)
- 快速计数: <10 ms

存储结构:

UltraLight 卡共 512 位，分为 16 页，每页为 4 个字节。存储结构如下：

页号	字节 0	字节 1	字节 2	字节 3	说明
0	SN0	SN1	SN2	BCC0	Serial Number
1	SN3	SN4	SN5	SN6	Serial Number
2	BCC1	保留	Lock0	Lock1	保留/Lock
3	OTP0	OTP1	OTP2	OTP3	OTP
4	Data0	Data1	Data2	Data3	Data read/write
5	Data4	Data5	Data6	Data7	Data read/write
6	Data8	Data9	Data10	Data11	Data read/write
7	Data12	Data13	Data14	Data15	Data read/write
8	Data16	Data17	Data18	Data19	Data read/write
9	Data20	Data21	Data22	Data23	Data read/write
10	Data24	Data25	Data26	Data27	Data read/write
11	Data28	Data29	Data30	Data31	Data read/write
12	Data32	Data33	Data34	Data35	Data read/write
13	Data36	Data37	Data38	Data39	Data read/write
14	Data40	Data41	Data42	Data43	Data read/write
15	Data44	Data45	Data46	Data47	Data read/write

(1) 第 0、1 页存放着卡的序列号等信息，只可读。依据 ISO/IEC14443-3 校验位计算如下：

BCC0=CT⊕SN0⊕SN1⊕SN2

BCC1=SN3⊕SN4⊕SN5⊕SN6

（2） 第 2 页为 LOCK BYTES，设置字节 2 和字节 3 对应的位可以将从第 3 页到 15 页单独地锁定为只读区域。

Lock0

L 7	L 6	L 5	L 4	L OTP	BL 15-10	BL 9-4	BL OTP

Lock1

L 15	L 14	L 13	L 12	L 11	L 10	L 9	L 8

Lx 锁定 X 页为只读

BLX 锁定对应的 Lx 位

注意：一旦 block-locking(BLX)位被设置为锁定配置，对应的内存区域将被冻结。如：BL15-10 设置为 1，则 L15 到 L10 再也不能改变。Lock0 和 Lock1 可以通过写命令来设置，写入的内容与当前内容进行位或操作得到新的内容，初始值为 0。该过程是不可逆转的。如果有一个位被置为 1，就再也不能置为 0。

（3） 第 3 页为 OTP，即一次性编程，初始值为 0。可以通过写命令来改变它的值，写入的值和当前值进行位或操作得到新的值。这个过程是不可逆转的。如果一个位被置为 1，将再也不能置回 0。

注意：该内存区域可以用作最大值为 32 的一次性计数器。

（4） 第 4 到 15 页为用户读/写区域，初始值为 0。

函数说明：

UltraLight 卡是一种单程票非接触式 IC 卡。

1、 卡片指令

reset--->request--->anticoll--->select--->read、write--->halt

2、 函数特别说明

UltraLight 卡操作函数同 Mifare One，有以下几点需要说明：

- 由于 UltraLight 卡的序列号为 7 个字节，所以防冲突函数不能够返回全部的卡片序列号，如要取得全部的卡片序列号请调用 rf_get_snr 函数，该函数为 UltraLight 卡专用函数。
- UltraLight 卡没有密码，故不需要装载密码，也不存在认证指令。
- Rf_read 函数返回 16 个字节的数据（即 4 个 page），故用户给的缓冲区必须大于 16 个字节。
- Rf_write 函数写入 16 个字节的数据，实际只有前面 4 个字节的数据写入指定的地址，其余字节可以补零。
- UltraLight 卡不存在增值、减值指令。
- 不支持高级函数。

功能介绍:

1. **REQA/Wake-Up:** 将卡片放入读写范围内，使它进入 ready 状态。Mifare Light 在 Idle 状态只接受 REQA 命令，在 Idle 和 Halt 状态.只接受 WAKE-UP 命令。
2. **ANTICOLLISION AND SELECT:** 这两个命令用在防冲突循环中。防冲突命令获取系列号部分，选卡命令用这个系列号从读写范围中的多张卡片中选择一张卡片。
3. **READ:** 根据页地址读出16个字节的内容。可以循环读取数据，例：如果页地址是

14，那么将读出14，15，0，1页的内容。

4. **WRITE:** 写卡命令用来对2页的锁字节，3页的 OTP 字节，4页到15页的数据字节进行操作。一个写卡命令完成一页的操作是指对一行共4个字节的操作。
5. **HALT:** 暂停命令用来将已操作完的卡设置进入等待状态（以 Halt 状态代替了 Idle 状态），这样就可以把已进行过防冲突处理，已知 UID 的卡片简单区别开来。这种机制是使读写器能在读写区域寻找到所有卡片的一个很有效的方法。

3 MIFARE STANDARD 4K

特性:

- 容量为4K 字节 EEPROM, 分为40个扇区，其中32个扇区为4个块，8个扇区为16个块，每块16个字节,以块为存取单位
- 每个扇区有独立的一组密码及访问控制
- 三重密码验证
- 每张卡有唯一序列号，为32位
- 具有防冲突机制，支持多卡操作
- 工作频率：13.56MHZ
- 完整的数据格式: 16 位 CRC, 奇偶校验,位编码,位计数
- 典型交易过程: <100 ms(包括备份管理)
- 非接触传送数据和无源（卡中无电源）
- 读写距离：在100mm 以内（与天线有关）
- 数据可保留10年
- 可循环改写100,000次

存储结构：

Mifare 4KByte 卡分为 40 个扇区，第 0 到 31 扇区每个扇区 4 块（块 0～3），第 32 到 39 扇区每个扇区 16 块，共 256 个块。按块号编址为 0～255。第 0 扇区的块 0（即绝对地址 0 块）用于存放厂商代码，已经固化，不可更改。前 32 个扇区的块 0、块 1、块 2 为**数据块**，用于存贮数据；块 3 为**控制块**，存放密码 A、存取控制、密码 B。后 8 个扇区的块 0 到块 14 为数据块，块 15 为控制块。控制块的结构如下：

A0A1A2A3A4A5	FF 07 80 69	B0B1B2B3B4B5
密码 A(6 字节)	存取控制(4 字节)	密码 B(6 字节)

控制属性

每个扇区的密码和存取控制都是独立的，可以根据实际需要设定各自的密码及存取控制。在**存取控制**中每个块都有相应的三个**控制位**,定义如下：

块 0：	C10	C20	C30
块 1：	C11	C21	C31
块 2：	C12	C22	C32

块 3： C13 C23 C33

三个控制位以正和反两种形式存在于存取控制字节中，决定了该块的访问权限（如进行减值操作必须验证 KEY A，进行加值操作必须验证 KEY B，等等）。三个控制位在存取控制字节中的位置如下（字节 9 为备用字节，默认值为 0x69）：

	bit 7	6	5	4	3	2	1	0
字节 6	C23_b	C22_b	C21_b	C20_b	C13_b	C12_b	C11_b	C10_b
字节 7	C13	C12	C11	C10	C33_b	C32_b	C31_b	C30_b
字节 8	C33	C32	C31	C30	C23	C22	C21	C20

（注： _b 表示取反）

其中，黑色区控制**块 3**，蓝色区控制**块 2**，绿色区控制**块 1**，红色区控制**块 0**。

数据块（块 0、块 1、块 2）的存取控制如下：

控制位(X=0..2)			**访 问 条 件 （对块 0、1、2）**			
C1X	**C2X**	**C3X**	**Read**	**Write**	**Increment**	**Decrement transfer restore**
0	0	0	KeyA|B	KeyA|B	KeyA|B	KeyA|B
0	1	0	KeyA|B	Never	Never	Never
1	0	0	KeyA|B	KeyB	Never	Never
1	1	0	KeyA|B	KeyB	KeyB	KeyA|B
0	0	1	KeyA|B	Never	Never	KeyA|B
0	1	1	KeyB	KeyB	Never	Never
1	0	1	KeyB	Never	Never	Never
1	1	1	Never	Never	Never	Never

（KeyA|B 表示密码 A 或密码 B，Never 表示任何条件下不能实现）

例如：当块 0 的存取控制位 C10 C20 C30=100 时，验证密码 A 或密码 B 正确后可读；验证密码 B 正确后可写；不能进行加值、减值操作。

控制块（块 3）的存取控制与**数据块**（块 0、1、2）不同，它的存取控制如下：

控制位			**密码 A**		**存取控制**		**密码 B**	
C13	**C23**	**C33**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**
0	0	0	Never	KeyA|B	KeyA|B	Never	KeyA|B	KeyA|B
0	1	0	Never	Never	KeyA|B	Never	KeyA|B	Never

1	0	0	Never	KeyB	KeyA|B	Never	Never	KeyB
1	1	0	Never	Never	KeyA|B	Never	Never	Never
0	0	1	Never	KeyA|B	KeyA|B	KeyA|B	KeyA|B	KeyA|B
0	1	1	Never	KeyB	KeyA|B	KeyB	Never	KeyB
1	0	1	Never	Never	KeyA|B	KeyB	Never	Never
1	1	1	Never	Never	KeyA|B	Never	Never	Never

例如：当块 3 的存取控制位 C13 C23 C33=100 时，表示：

密码 A： 不可读，验证 KEYB 正确后，可写（更改）。

存取控制：验证 KEYA 或 KEYB 正确后，可读不可写。

密码 B： 不可读，验证 KEYB 正确后，可写。

工作原理

卡片的电气部分只由一个天线和 ASIC 组成。

天线：卡片的天线是只有几组绕线的线圈，很适于封装到 IS0 卡片中。

ASIC :卡片的 ASIC 由一个高速（106KB 波特率）的 RF 接口，一个控制单元和一个 8K 位 EEPROM 组成。

读写器向 M1 卡发一组固定频率的电磁波，卡片内有一个 LC 串联谐振电路，其频率与讯写器发射的频率相同，在电磁波的激励下，LC 谐振电路产生共振，从而使电容内有了电荷，在这个电容的另一端，接有一个单向导通的电子泵，将电容内的电荷送到另一个电容内储存，当所积累的电荷达到 2V 时，此电容可做为电源为其它电路提供工作电压，将卡内数据发射出去或接取读写器的数据。

注意事项：

1) 由于读写器只能保存一套 16 个扇区的密码，所以扇区号大于 15 扇区的密码必须重新装载，同时调用认证函数 rf_authentication_2( )。

例如：

```
int    sector=16,mode=0;
HANDLE icdev;
Unsigned data[17];
rf_authentication_2(icdev, mode, sector%16, sector*4);
rf_read(icdev, sector*4, data);
```

2) 值操作中数据块的长度为 4 字节，存储结构如下：

字节号	0 1 2 3	4 5 6 7	8 9 10 11	12	13	14	15
描述	Value	Value_b	Value	Addr	Addr_b	Addr	Addr_b

注（_b：表示取反）

功能介绍：

1. **Request Standard/ALL**: 一张卡上电复位后就可以响应读写器发出的寻卡命令。读写器向天线范围内的所有卡片发出命令，并识别卡片的型号。

2. **Anticollision Loop :**在防冲突循环中将读出卡片的系列号。如果有几张卡片都在读写器的读写范围内，可以通过唯一的系列号区别它们，并选中其中一张卡片，其余没有选中的卡片将进入等待状态，等待下一次寻卡命令。
3. **Select Card:** 用选卡命令读写器选择一张卡片来进行密码验证和有关的存储操作。卡片将对代码为08h 的选卡命令 ATS 做出响应，该命令决定了所选卡片的类型。
4. **Pass Authentication:** 读写器选中一张卡片后就指定了后续存取访问的存储空间，并以次响应开始3重密码验证。
5. **HALT:** 调用 rf_halt () 函数来停止对卡片的所有操作，卡片进入 HALT 状态。
6. **Memory Operations:** 密码验证通过后，可进行以下操作：
    - 读块: 读取一个存储块的内容
    - 写块: 写入一个存储块的内容.
    - 减值:减少一个块的值并保存在内部寄存器内
    - 增值: 增加一个块的值并保存在内部寄存器内
    - 保存: 将块的内容写入数据寄存器中
    - 传输: 将内部寄存器的内容写入某一块中

    注:用 rf_ restore () 函数将某一块的内容传到内部寄存器内，然后用 rf_truansfer() 函数将内部寄存器的内容传到卡片其余块中。通过这种方法，可以将数据从一块传向另一块。

4 MIFARE LIGHT 卡介绍

一、概述

MIFARE LIGHT 卡是一种小容量卡，共 384 位，适合于一卡一用。

二、主要指标

- 容量为 384 位
- 16 位的数值计算
- 128 位的数据区（如果不用钱包文件可达 192 位）
- 用户可自定义控制权限
- 唯一的 32 位序列号
- 工作频率：13.56MHZ
- 通信速率：106KB 波特率
- 防 冲 突：同一时间可处理多张卡
- 读写距离：在 10cm 以内（与天线有关）
- 卡内无需电源

三、存储结构

ML 卡共 384 位，分为 12 页，每页为 4 个字节。存储结构如下：

页号	字节 0	字节 1	字节 2	字节 3	

0	SerNr(0)	SerNr(1)	SerNr(2)	SerNr(3)	Block 0
1	SerNr(4)	Size Code	Type(0)	Type(1)	
2	Data(0)	Data(1)	Data(2)	Data(3)	Data1
3	Data(4)	Data(5)	Data(6)	Data(7)	
4	Value(0)	Value(1)	Value_b(0)	Value_b(1)	Value
5	Value(0)	Value(1)	Value_b(0)	Value_b(1)	
6	KeyA(0)	KeyA(1)	KeyA(2)	KeyA(3)	KeyA
7	KeyA(4)	KeyA(5)	AC-A	AC-A_b	
8	KeyB(0)	KeyB(1)	KeyB(2)	KeyB(3)	KeyB
9	KeyB(4)	KeyB(5)	AC-B	AC-B_b	
A	Data(0)	Data(1)	Data(2)	Data(3)	Data2
B	Data(4)	Data(5)	Data(6)	Data(7)	

（ 注：_b 表示取反 ）

1) 第 0、1 页存放着卡的序列号等信息，只可读。

2) 第 2、3 页及 A、B 两页数据块，可存贮一般的数据。

3) 和 4、5 页为数值块，可作为钱包使用，两字节的值以正和反两种形式存贮。只有减值操作，没有加值操作。如果不做钱包使用，则可以做为普通的数据块使用。

4）第 6、7、8、9 页存储着密码 A（6 字节）、密码 B（6 字节）及存取控制。

5）第 7 页的 2 字节、第 9 页的 2 字节为存储控制，存储控制以正和反的形式存两次。

Bit 7	---
Bit 6	---
Bit 5	Data2—Write –Enable
Bit 4	Data2—Read—Enable
Bit 3	Key+AC—Write—Enable
Bit 2	Value—Write—Enable
Bit 1	Data1—Write—Enable
Bit 0	Data1—Read—Enable

例如：AC-A 的初始值为 ff，即‘ 11111111 ’，即：

Data1：可读、可写；

Value：可写；

AC-A：可写；

Data2：可读、可写；

6）一次写一页（4 个字节），一次读两页（8 个字节）。