AirKiss 库文件使用指南

WIFI 模组如何实现 AirKiss、内网发现

发布时间：2015-08-11
版本：2.2

版本记录

作者	发布日期	版本	备注
Zorrowu	2015-02-05	0.1	初稿，对应 AirKiss 库版本 0.8
Zorrowu	2015-02-09	0.2	添加资源占用详情，增加路由器测试结果
Zorrowu	2015-02-12	1.0	确定发布版本，增加 airkiss_version()接口
Zorrowu	2015-03-31	1.1	完善文档，添加流程图，Q&A
Zorrowu	2015-06-08	1.2	更新 Q&A
Zorrowu	2015-06-30	2.0	增加解析手机数据包的算法和微信内网协议说明，切换信道后增加清缓存接口
Oscarxie	2015-07-02	2.1	文档格式、样式更新
Zorrowu	2015-08-11	2.2	airkiss_change_channel 参数说明错误修订

评审记录

评审人	评审时间	评审内容	评审意见
Zorro	2015-07-02	全部文档	

目标读者

WIFI 芯片开发人员，WIFI 模块调试人员，微信内网发现、近场扫描开发人员

目录

1. 什么是 AirKiss™技术

AirKiss 是微信硬件平台提供的一种 WIFI 设备快速入网配置技术 ,要使用微信客户端的方式配置设备入网，需要设备支持 AirKiss 技术。目前已经有越来越多的芯片和模块厂商，提供了支持 AirKiss 技术的方案。

1.1 AirKiss 技术使用场景

AirKiss 主要在如下场景中使用：

1. 待接入互联网的设备不具备输入输出能力，如空调、空气净化器、烟雾报警器等。
2. 用户不具备通过设备热点的方式进行配置的能力，如老人、家庭主妇等缺乏相关 IT 知识的用户人群。

1.2 AirKiss 技术应用实例

以智能插座为例，下文将说明 AirKiss 技术的应用方案和交互流程。智能插座属于物联网智能控制类设备，它可用于家电(比如电灯、热水器等)的智能化开关控制。智能插座的特点是小型化且低功耗，显而易见，该设备并不适合于配置屏幕与键盘等输入外设。在这种情况下，AirKiss 技术能完美解决其 SSID 与密码的传输、设置问题。

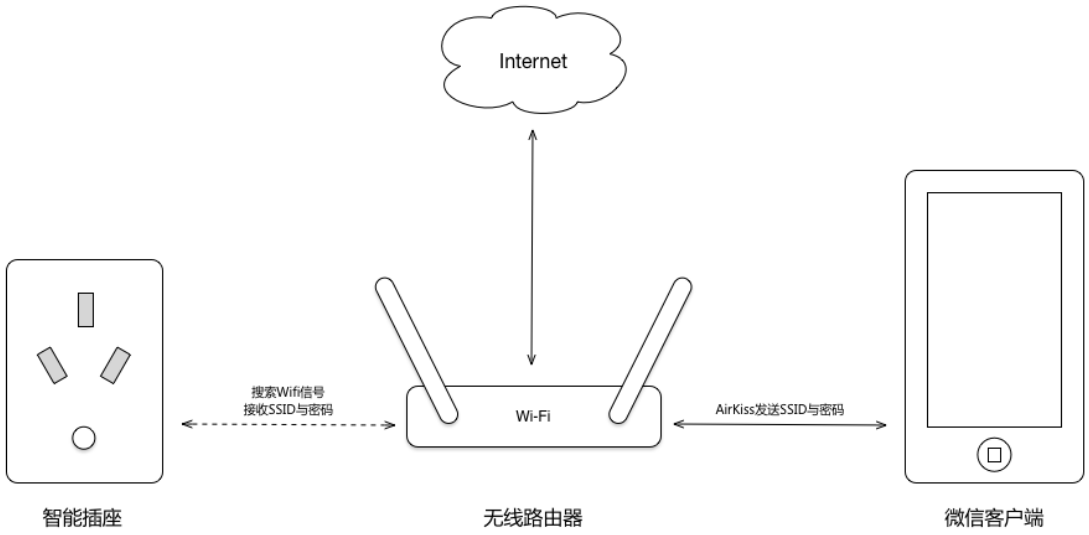

图 1 AirKiss 配置设备联网示意图

AirKiss 技术对应用设备的硬件几乎没有额外的要求，配置时需要设备能够进入 AirKiss 模式。在本例中，智能插座在按下了配置按键之后，指示灯闪烁进入 AirKiss 模式，成为了 AirKiss 技术中信息的接收方。用户则可以使用微信手机客户端，成为 AirKiss 技术中的信息发送方。

图 2 微信中使用 AirKiss 操作界面

用户使用 Air Kiss 的交互流程如下：

1. 用户按下智能插座上的配置按键，AirKiss 指示灯闪烁，智能插座进入信息接收状态。
2. 用户打开微信手机客户端，进入设备的联网配置界面（设备厂商开发的 HTML5 页面），唤起 AirKiss 的 SSID 与密码发送界面，当前无线网络环境下无线路由器的 SSID 已经默认选中，用户只需要填写密码，然后点击发送即可。

整个 AirKiss 过程将在 15 秒内完成。

1.3 微信内网发现协议的应用场景

为了提供更好的用户体验 ,微信制定了一套内网发现协议并集成到了微信客户端内，只要设备实现了该内网协议并运行，就能被微信客户端所发现和绑定。AirKiss 库目前已经集成了该内网协议，开发者可以不需要了解内网协议的详细细节，只需要按照下文所描述的流程调用即可。

1.4 AirKiss 技术的优势

相比其它配置方式，AirKiss 技术有着以下几个显著的优势：

1. 用户可以使用最为熟悉的微信客户端来操作入网配置 ,无需下载额外的第三方软件即可。
2. 用户无需首先将设备配置为热点模式并连接 ,在配置模式下可直接将无线路由器的 SSID 发送至设备。
3. 信息传输安全，AirKiss 支持 AES-128 加密，密钥可以由厂家自行设定。

2. AirKiss 库组成结构介绍

2.1 AirKiss 库文件组成

AirKiss 库由以下文件组成：

`airkiss.h`：

AirKiss 库头文件，要使用 AirKiss 功能必须包含该头文件，主要定义了相关参数结构体及 AirKiss 库函数接口，AES 加密功能启用宏也在该文件中。

`libairkiss.a`：

不带 AES 加密功能的 AirKiss 静态库文件，如果厂家不需要 AES 加密功能则可使用本静态库，占用的资源比带 AES 功能的静态库更少，使用本静态库记得关闭 `airkiss.h` 中的 AES 加密功能宏。

`libairkiss_aes.a`：

支持 AES 加密功能的 AirKiss 静态库文件，如果厂家需要使用 AES 加密功能则需要链接本静态库，占用的资源相对 `libairkiss.a` 较多，使用本静态库记得开启 `airkiss.h` 中的 AES 加密功能宏。

`libairkiss_log.a`：

在 `libairkiss.a` 的基础上添加了 log 打印，可以用于 debug。

`libairkiss_aes_log.a`：

在 `libairkiss_aes.a` 的基础上添加了 log 打印，可以用于 debug。

注意：`airkiss.h` 需要添加到头文件路径中，`libairkiss.a` 和 `libairkiss_aes.a` 只需要使用其中一个，根据具体需要选择，并设置好 `airkiss.h` 文件中的 AES 加密功能宏，出于安全考虑，建议使用 AES 加密功能的 AirKiss 静态库。

3. AirKiss 库使用流程说明

3.1 设备平台能力要求

无法满足以下软件及硬件能力要求的设备或模块将无法使用 AirKiss 功能。

硬件能力要求：

1、 能够切换信道；

2、 具备定时器功能，能够提供 100ms 的定时中断；

3、 能够设置为混杂模式，接收 802.11 网络帧；

4、 提供一种进入 AirKiss 模式的控制方式，例如一个按键；

软件能力要求：

1、 能够提供类似标准 memset 函数的功能函数；

2、 能够提供类似标准 memcpy 函数的功能函数；

3、 能够提供类似标准 memcmp 函数的功能函数；

4、 能够提供至少 232 字节的全局缓冲空间(完成 AirKiss 后用户可用于自己的应用程序或进行释放)；

5、 带 AES 功能的静态库文件大小为 32KB，不带 AES 功能的静态库文件大小为 13KB，实际链接以后占用资源不同，以 ESP8266 平台为例，实现不带 AES 功能的 AirKiss 要占用 2304 字节，实现带有 AES 功能要占用 5456 字节。以上统计包括实现 AirKiss 功能外部逻辑函数代码。

3.2 AirKiss 库使用说明

使用 AirKiss 库实现 AirKiss 功能的流程如下（`#include "airkiss.h"`）：

1、 创建 AirKiss 全局缓冲区：`airkiss_context_t akcontex;`如果平台支持 `malloc` 等动态内存申请，也可以通过动态申请的方式申请空间，完成 AirKiss 流程或超时（超时时长用户可以自定，建议 30～40s）以后进行释放；

2、 为 AirKiss 库配置与平台相关的接口函数结构体，可以为静态 `const` 类型，下面示例中的函数都为标准 C 库中对应的函数名，如果平台没有该函数需要把平台实现相同功能的函数名填上，其中最后一项为打印函数，可以填 0，其他为必填项，否则调用 `airkiss_init()` 接口会返回失败：

```
const airkiss_config_t akconf = {
(airkiss_memset_fn)&memset,
(airkiss_memcpy_fn)&memcpy,
(airkiss_memcmp_fn)&memcmp,
(airkiss_printf_fn)&printf };
```

3、 调用 AirKiss 初始化接口，接口参数为前两步创建的变量地址：

```
ret = airkiss_init(&akcontex, &akconf);
```

如果 ret 返回值小于 0 则表示初始化失败，通常为参数错误，等于 0 为成功，如果用户在一次 AirKiss 的流程中想重新开始新流程，需要通过调用该接口实现。

4、 如果 `airkiss.h` 文件中开启了 AES 的宏并且链接了 `libairkiss_aes.a` 静态库，则需要调用设置密钥接口， key 可以为局部变量：

```
airkiss_set_key(&akcontex, key, strlen(key));
```

```
/*
 * 定义AIRKISS_ENABLE_CRYPT为1以启用AirKiss加密功能
 */
#ifndef AIRKISS_ENABLE_CRYPT
#define AIRKISS_ENABLE_CRYPT 1
#endif
```

图 3 代码示例

5、 完成以上初始化流程以后就可以开启 100ms 定时器，在定时中断函数中依次切换信道；

6、 设置模块为混杂模式，接收 802.11 网络帧，每收到一帧数据，将数据起始指针和数据长度传递给 `airkiss_recv()` 接口，并判断该接口的返回值，

如果返回 AIRKISS_STATUS_CHANNEL_LOCKED 则表示信道已经锁定了，需要关闭定时器停止切换信道，如果返回 AIRKISS_STATUS_COMPLETE 则表示 AirKiss 完成可以调用 airkiss_get_result()接口读取参数，其他值可以不用进行处理，具体实现可以参考下一章节的使用示例。

3.3 AirKiss 库调用流程图

设备端调用 AirKiss 库实现 AirKiss 功能的流程图如下所示：

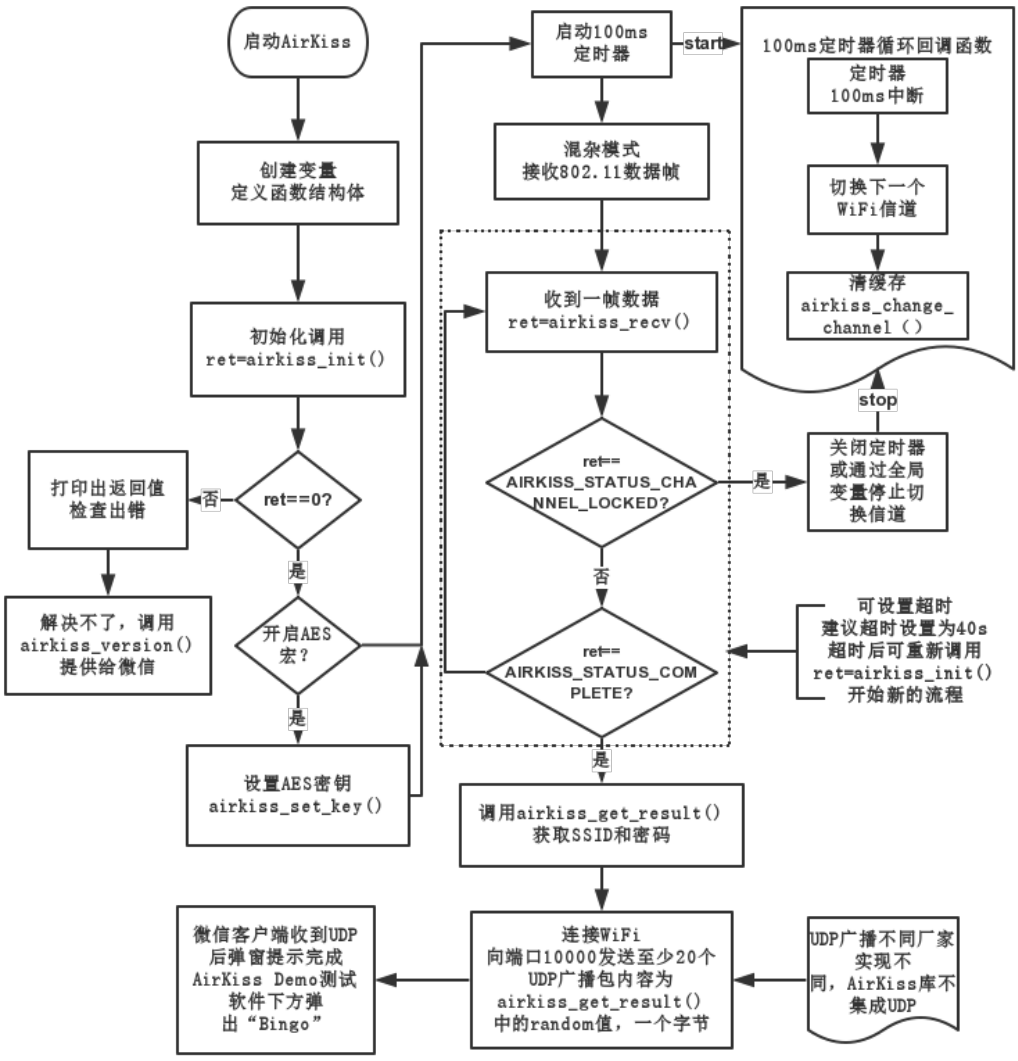

图 4 AirKiss 工作流程示意图

3.4 AirKiss 库内网发现功能使用说明

微信内网发现功能与 AirKiss 是两个相对独立的模块，两者没有依赖关系，使用 AirKiss 库实现微信内网发现功能的流程如下（#include "airkiss.h"）：

1、 为 AirKiss 库创建与平台相关的接口函数结构体，可以为静态 const 类型，下面示例中的函数都为标准 C 库中对应的函数名，如果平台没有该函数需要把平台实现相同功能的函数名填上，其中最后一项为打印函数，可以填 0，其他为必填项，也可以复用在 AirKiss 流程中创建的变量：

```
const airkiss_config_t akconf = {
(airkiss_memset_fn)&memset,
(airkiss_memcpy_fn)&memcpy,
(airkiss_memcmp_fn)&memcmp,
(airkiss_printf_fn)&printf };
```

2、 根据使用的平台自行创建 UDP，对 12476 端口进行监听，ip 地址不限制；

3、 将从 12476 端口接收到的数据包传给 airkiss_lan_recv(const void* body, unsigned short length, const airkiss_config_t* config)函数，body 为 UDP 消息体数据的起始指针，length 为数据的有效长度，config 为第 1 步创建的变量，函数的返回值在头文件中进行了定义，这里暂时只需要处理返回 AIRKISS_LAN_SSDP_REQ 的情况，对于其他值用户可以选择打印出来方便调试；

4、 在上一步接收到 AIRKISS_LAN_SSDP_REQ 结果后，设备需要向数据包的发送发回复响应包，即以数据包的源 IP 和源端口为目的 IP 和目的端口，响应包可以通过调用 airkiss_lan_pack(airkiss_lan_cmdid_t ak_lan_cmdid, void* appid, void* deviceid, void* _datain, unsigned short inlength, void* _dataout, unsigned short* outlength, const airkiss_config_t* config);函数实现打包，ak_lan_cmdid 为要打包的类型，appid 为厂商公众号 ID，deviceid 为设备 ID，_datain 为要发送的数据，inlength 为发送数据的长度，_dataout 为打包后的数据缓冲区，outlength 为缓冲区的空间，函数成功返回后将赋值为数据包的实际长度，config 为第 1 步创建的变量，示例：airkiss_lan_pack(AIRKISS_LAN_SSDP_RESP_CMD, "gh_27098xx","BD5D7xx", 0, 0, lan_buf, &lan_buf_len, &akconf);因为响应包不需要其他数据，所以输入数据和长度都设置为 0；

5、 airkiss_lan_pack返回AIRKISS_LAN_PAKE_READY后表明数据包打包完成了，数据存放在第 4 步示例中的 lan_buf 中，有效数据长度为 lan_buf_len，接下来用户根据使用的平台将数据通过 UDP 的方式发送给对方即可。

6、 通过前面的步骤实现的是一问一答的服务发现模式，设备也可以直接向网络发送上线通知数据包，无需等待前面提到的请求包。上线通知包的目的 IP 为 255.255.255.255，目的端口为 12476，消息包可以通过调用 airkiss_lan_pack() 函数生成，示例：airkiss_lan_pack(AIRKISS_LAN_SSDP_NOTIFY_CMD, "gh_27098xx","BD5D7xx", 0, 0, lan_buf, &lan_buf_len, &akconf);函数返回 AIRKISS_LAN_PAKE_READY 后将数据通过 UDP 的方式发送到网络上即可。

3.5 AirKiss 库内网发现功能调用流程图

设备端调用 AirKiss 库实现内网发现功能的流程图如下所示：

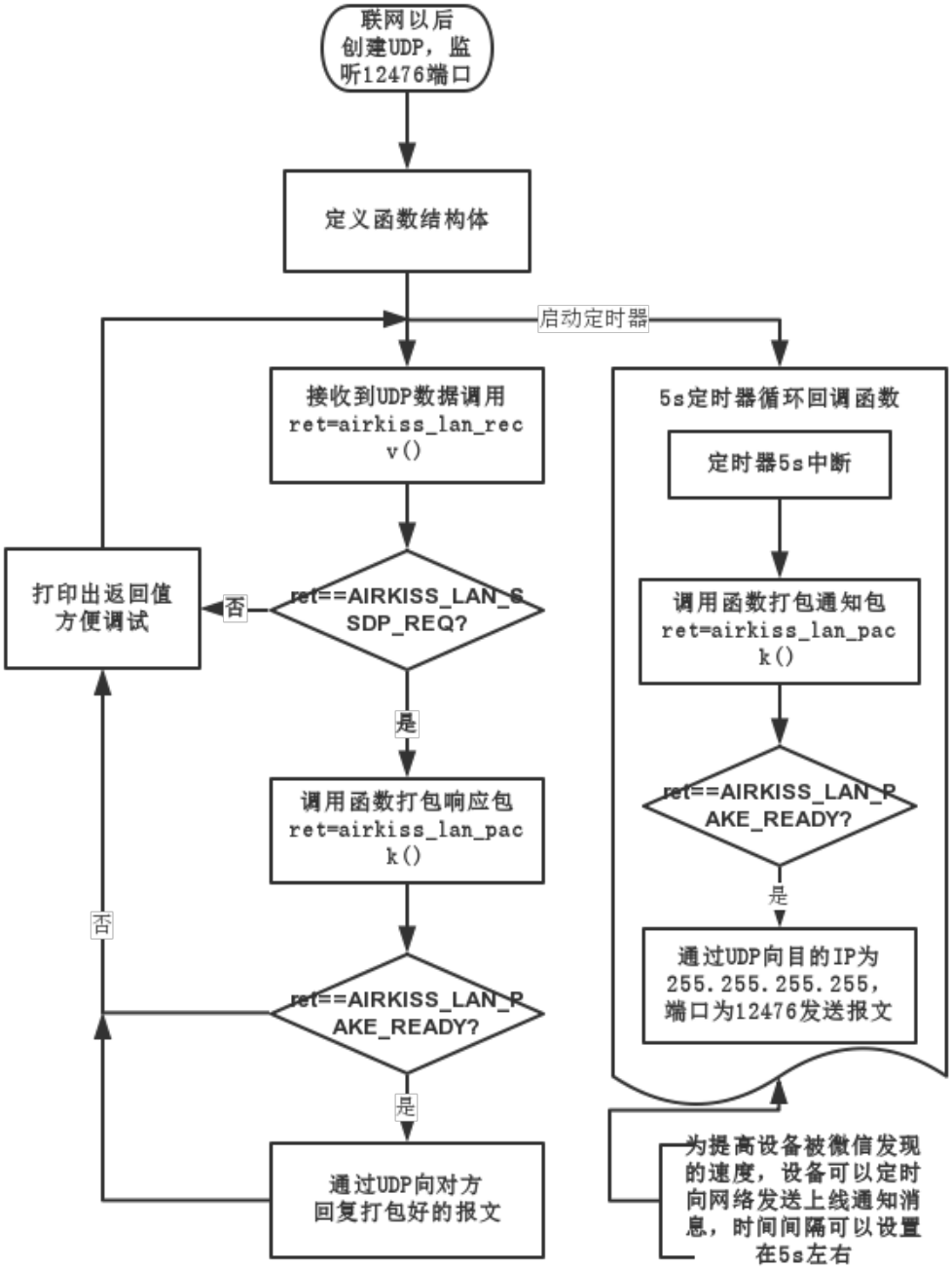

图 5 内网发现工作流程示意图

4. AirKiss 库使用示例

本小节将以 ESP8266 为示例平台，介绍如何利用 AirKiss 库实现 AirKiss 功能，完整的代码如下：

```
//平台相关头文件
#include "ets_sys.h"
#include "driver/uart.h"
#include "osapi.h"
#include "ip_addr.h"
#include "user_interface.h"

//包含AirKiss头文件
#include "airkiss.h"

//当前监听的无线信道
uint8_t cur_channel = 1;

//用于切换信道的定时器，平台相关
os_timer_t time_serv;

//AirKiss过程中需要的RAM资源，完成AirKiss后可以供其他代码使用
airkiss_context_t akcontex;

//另一种更节省资源的使用方法，通过malloc动态申请RAM资源，完成后利用free释放，需要平台支持
//示例：
//airkiss_context_t *akcontexprt;
//akcontexprt =
//(airkiss_context_t
*)os_malloc(sizeof(airkiss_context_t));
```

```
//定义AirKiss库需要用到的一些标准函数，由对应的硬件平台提供，前三个为必要函数
const airkiss_config_t akconf =
{
    (airkiss_memset_fn)&memset,
    (airkiss_memcpy_fn)&memcpy,
    (airkiss_memcmp_fn)&memcmp,
    0
};

/*
 * 平台相关定时器中断处理函数，100ms中断后切换信道
 */
static void time_callback(void)
{
    //切换信道
    if (cur_channel >= 13)
        cur_channel = 1;
    else
        cur_channel++;
    wifi_set_channel(cur_channel);
    airkiss_change_channel(&akcontex);//清缓存
}

/*
 * airkiss成功后读取配置信息，平台无关，修改打印函数即可
 */
static void airkiss_finish(void)
{
    int8_t err;
    uint8 buffer[256];
    airkiss_result_t result;
    err = airkiss_get_result(&akcontex, &result);
```

```
    if (err == 0)
    {
        uart0_sendStr("airkiss_get_result() ok!");
        os_sprintf(buffer,
            "ssid = \"%s\", pwd = \"%s\", ssid_length = %d,
            "pwd_length = %d, random = 0x%02x\r\n",
            result.ssid, result.pwd, result.ssid_length,
            result.pwd_length, result.random);
        uart0_sendStr(buffer);
    }
    else
    {
        uart0_sendStr("airkiss_get_result() failed !\r\n");
    }
}

/*
 * 混杂模式下抓到的802.11网络帧及长度，平台相关
 */
static void wifi_promiscuous_rx(uint8 *buf, uint16 len)
{
    int8_t ret;
    //将网络帧传入airkiss库进行处理
    ret = airkiss_recv(&akcontex, buf, len);
    //判断返回值，确定是否锁定信道或者读取结果
    if ( ret == AIRKISS_STATUS_CHANNEL_LOCKED)
        os_timer_disarm(&time_serv);
    else if ( ret == AIRKISS_STATUS_COMPLETE )
    {
        airkiss_finish();
        wifi_promiscuous_enable(0);//关闭混杂模式，平台相关
    }
}
```

```
/*
 * 初始化并开始进入AirKiss流程，平台相关
 */
void start_airkiss(void)
{
    int8_t ret;
    //如果有开启AES功能，定义AES密码，注意与手机端的密码一致
    const char* key = "Wechatiothardwav";

    uart0_sendStr("Start airkiss!\r\n");
    //调用接口初始化AirKiss流程，每次调用该接口，流程重新开始，akconf需要预先设置好参数
    ret = airkiss_init(&akcontex, &akconf);
    //判断返回值是否正确
    if (ret < 0)
    {
        uart0_sendStr("Airkiss init failed!\r\n");
        return;
    }

#if AIRKISS_ENABLE_CRYPT
    //如果使用AES加密功能需要设置好AES密钥，注意包含正确的库文件，头文件中的宏要打开
    airkiss_set_key(&akcontex, key, strlen(key));
#endif
    uart0_sendStr("Finish init airkiss!\r\n");
    //以下与硬件平台相关，设置模块为STATION模式并开启混杂模式，启动定时器用于定时切换信道
    wifi_station_disconnect();
    wifi_set_opmode(STATION_MODE);
    cur_channel = 1;
    wifi_set_channel(cur_channel);
```

```
    os_timer_setfn(&time_serv, (os_timer_func_t
*)time_callback, NULL);
    os_timer_arm(&time_serv, 100, 1);
    wifi_set_promiscuous_rx_cb(wifi_promiscuous_rx);
    wifi_promiscuous_enable(1);
}

/*
 * 硬件平台初始化，与AirKiss库使用无关
 */
void  ICACHE_FLASH_ATTR
user_init(void)
{
    uart_init(115200, 115200);
    system_init_done_cb(start_airkiss);
}
```

下面对上面的示例代码进行分析：

文件开头定义并初始化了一些与 AirKiss 功能相关的全局变量：

```
//包含AirKiss头文件
#include "airkiss.h"

//当前监听的无线信道
uint8_t cur_channel = 1;
//用于切换信道的定时器，平台相关
os_timer_t time_serv;
//AirKiss过程中需要的RAM资源，完成AirKiss后可以供其他代码使用
airkiss_context_t akcontex;
//另一种更节省资源的使用方法，通过malloc动态申请RAM资源，完成后利用free释放，需要平台支持
// airkiss_context_t *akcontexprt;
//示例: akcontexprt = (airkiss_context_t *)os_malloc(sizeof(airkiss_context_t));

//定义AirKiss库需要用到的一些标准函数，由对应的硬件平台提供，前三个为必要函数
const airkiss_config_t akconf = {
    (airkiss_memset_fn)&memset,
    (airkiss_memcpy_fn)&memcpy,
    (airkiss_memcmp_fn)&memcmp,
    0
};
```

图 6 代码示例

ESP8266 平台的入口函数为 `user_init(void)`，在该函数中只是初始了一下串口，并注册一个回调函数 `start_airkiss()`，SDK 完成初始化后会调用该函数。在该函数中将创建的变量的地址作为参数传递给了 `airkiss_init()`，并判断该函数的返回值是否正确，接着判断 `airkiss.h` 头文件中是否开启了

AES 加密功能，如果开启的话需要调用 airkiss_set_key()设置一下密钥，该密钥要跟手机等发送 AirKiss 数据的设备保持一致。到这里 AirKiss 库的初始化就完成，下面开始是跟平台相关的初始化，主要有两个任务：

一、设置一下要监听的信道，示例代码设置为信道 1，同时启动一个定时器，定时器周期为 100ms，定时时间到了以后会自动重载；

二、设置模块为 STATION 模式(AP 模式能够抓到包也可以)，并开启混杂模式，注册了一个混杂模式收包的回调接口：

```
/*
 * 初始化并开始进入AirKiss流程，平台相关
 */
void start_airkiss(void)
{
    int8_t ret;
    //如果有开启AES功能，定义AES密码，注意与手机端的密码一致
    const char* key = "Wechatiothardwav";

    uart0_sendStr("\r\nStart airkiss!\r\n");
    //调用接口初始化AirKiss流程，每次调用该接口，流程重新开始，akconf需要预先设置好参数
    ret = airkiss_init(&akcontex, &akconf);
    //判断返回值是否正确
    if ( ret < 0 ) {uart0_sendStr("Airkiss init failed!\r\n"); return;}
    #if AIRKISS_ENABLE_CRYPT
        //如果使用AES加密功能需要设置好AES密钥，注意包含正确的库文件，头文件中的宏要打开
        airkiss_set_key(&akcontex, key, strlen(key));
    #endif
    uart0_sendStr("Finish init airkiss!\r\n");
    //以下与硬件平台相关，设置模块为STATION模式并开启混杂模式，启动定时器用于定时切换信道
    wifi_station_disconnect();
    wifi_set_opmode(STATION_MODE);
    cur_channel = 1;
    wifi_set_channel(cur_channel);
    os_timer_setfn(&time_serv, (os_timer_func_t *)time_callback, NULL);
    os_timer_arm(&time_serv, 100, 1);
    wifi_set_promiscuous_rx_cb(wifi_promiscuous_rx);
    wifi_promiscuous_enable(1);
}
```

图 7 代码示例

定时器中断处理函数代码如下，每次进到中断函数判断一下当前信道是否超出范围，如果是则重置为第 1 个信道，否则累加：

```
/*
 * 平台相关定时器中断处理函数，100ms中断后切换信道 */
static void time_callback(void) {
    //切换信道
    if ( cur_channel >= 13)
        cur_channel = 1;
    else
        cur_channel++;
    wifi_set_channel(cur_channel);
    airkiss_change_channel(&akcontex);//清缓存
}
```

图 8 代码示例

收到 802.11 网络帧以后的回调函数处理代码如下，将收到的数据起始指针和长度传递给 airkiss_recv()函数，并判断该函数的返回值，如果返回

AIRKISS_STATUS_CHANNEL_LOCKED 则关闭定时器，停止切换信道，用户也可以通过一个开关变量来实现，如果返回 AIRKISS_STATUS_COMPLETE 则表示 AirKiss 已经完成，可以读取 SSID、密码和随机数了，本示例自定义了一个 airkiss_finish 函数来实现，读取完参数以后可以关闭混杂模式，如果用户没有关闭混杂模式，并且收到包以后继续调用 airkiss_recv()函数的话，该函数同样会返回 AIRKISS_STATUS_COMPLETE，除非用户重新调用一次 airkiss_init()函数：

```
/*
 * 混杂模式下抓到的802.11网络帧及长度，平台相关
 */
static void wifi_promiscuous_rx(uint8 *buf, uint16 len) {
    int8_t ret;
    //将网络帧传入airkiss库进行处理
    ret = airkiss_recv(&akcontex, buf, len);
    //判断返回值，确定是否锁定信道或者读取结果
    if ( ret == AIRKISS_STATUS_CHANNEL_LOCKED)  os_timer_disarm(&time_serv);
    else if ( ret == AIRKISS_STATUS_COMPLETE ) {
        airkiss_finish();
        wifi_promiscuous_enable(0);//关闭混杂模式，平台相关
    }
}
```

图 9 代码示例

自定义的 airkiss_finish()函数如下，在该函数内定义了一个局部变量 airkiss_result_t result，然后通过 airkiss_get_result()接口读取结果，如果返回成功，则通过平台提供的打印函数对数据进行序列化打印，最后输出到串口：

```
/*
 * airkiss成功后读取配置信息，平台无关，修改打印函数即可
 */
static void airkiss_finish(void){
    int8_t err;
    uint8 buffer[256];
    airkiss_result_t result;
    err = airkiss_get_result(&akcontex, &result);
    if (err == 0)
    {
        uart0_sendStr("-------------------- airkiss_get_result() ok!");
        os_sprintf(buffer, "ssid = \"%s\", pwd = \"%s\", ssid_length = %d, pwd_length
            = %d, random = 0x%02x\r\n",result.ssid, result.pwd, result.ssid_length,
            result.pwd_length, result.random);
        uart0_sendStr(buffer);
    }
    else
    {
        uart0_sendStr("-------------------- airkiss_get_result() failed !\r\n");
    }
}
```

图 10 代码示例

示例代码调试输出数据内容如下所示：

```
Start airkiss!
Finish init airkiss!
-------------------- airkiss_get_result() ok!ssid = "Tencent-GuestWiFi", pwd =
"[redacted]", ssid_length = 17, pwd_length = [redacted], random = 0x04
```

图 11 调试输出数据

模块拿到上面的数据以后还需要进行 AirKiss 流程的最后一步（详细内容可查看 AirKiss 技术实现方案文档）利用接收到的 SSID 和密码以后连上对应的路由器，立即发送以上面打印出来的 random 数为内容的 UDP 广播包(只有 1 个数据)，目的端口号为 10000，建议广播包的个数至少为 20 个，发送方收到该广播包后就能确认接收方已经准确接收到所有数据了 ,由于各平台连接路由器、发送 UDP 广播包的实现差异较大，并且该功能为模块自带功能，与 AirKiss 库无关，这里不进行举例说明。

4.1 内网发现功能实现示例

本小节将介绍如何利用 AirKiss 库实现微信内网发现功能 ,示例流程伪代码如下，这里假设设备已经获取到 SSID 和密码并且已经连接上路由器了：

```
//平台相关头文件
#include "ets_sys.h"
#include "driver/uart.h"
#include "osapi.h"
#include "ip_addr.h"
#include "user_interface.h"
#include "espconn.h"
#include "os_type.h"
//包含AirKiss头文件
#include "airkiss.h"
//用于发送上线通知包的定时器，平台相关
os_timer_t time_serv;
//UDP套接字相关变量，平台相关
esp_udp airkiss_udp;
struct espconn ptrairudpconn;

//用于缓存回包的数据缓冲区，也可以为局部变量
uint8_t lan_buf[200];
uint16_t lan_buf_len;
```

```
#define DEVICE_TYPE   "gh_27098xxxxx"
#define DEVICE_ID     "BD5D7899xxxxx"
#define DEFAULT_LAN_PORT    12476
//定义AirKiss库需要用到的一些标准函数，由对应的硬件平台提供，前三个为必要函数
const airkiss_config_t akconf =
{
    (airkiss_memset_fn)&memset,
    (airkiss_memcpy_fn)&memcpy,
    (airkiss_memcmp_fn)&memcmp,
    0
};
/*
 * 平台相关定时器中断处理函数，比较正确的做法是在中断里面发送信号通知任务发送，这里
 * 为了方便说明直接发送
 */
static void time_callback(void)
{
    airkiss_udp.remote_port = DEFAULT_LAN_PORT;
    airkiss_udp.remote_ip[0] = 255;
    airkiss_udp.remote_ip[1] = 255;
    airkiss_udp.remote_ip[2] = 255;
    airkiss_udp.remote_ip[3] = 255;
    lan_buf_len = sizeof(lan_buf);
    ret = airkiss_lan_pack(AIRKISS_LAN_SSDP_NOTIFY_CMD,
DEVICE_TYPE, DEVICE_ID, 0, 0, lan_buf, &lan_buf_len, &akconf);
    if (ret != AIRKISS_LAN_PAKE_READY) {
        uart0_sendStr("Pack lan packet error!");
        return;
    }
    ret = espconn_sent(&ptrairudpconn, lan_buf, lan_buf_len);
    if (ret != 0)  {
        uart0_sendStr("UDP send error!");
```

```
    }
    uart0_sendStr("Finish send notify!");
}
/*
 * 硬件平台相关，UDP监听端口数据接收处理函数
 */
void wifilan_recv_callbk(void *arg, char *pdata, unsigned short
len)
{
    airkiss_lan_ret_t ret = airkiss_lan_recv(pdata, len, &akconf);
    airkiss_lan_ret_t packret;
    switch (ret){
        case AIRKISS_LAN_SSDP_REQ:
        airkiss_udp.remote_port = DEFAULT_LAN_PORT;
        lan_buf_len = sizeof(lan_buf);
        packret = airkiss_lan_pack(AIRKISS_LAN_SSDP_RESP_CMD,
    DEVICE_TYPE, DEVICE_ID, 0, 0, lan_buf, &lan_buf_len, &akconf);
        if (packret != AIRKISS_LAN_PAKE_READY) {
            uart0_sendStr("Pack lan packet error!");
            return;
        }
        packret = espconn_sent(&ptrairudpconn, lan_buf,
        lan_buf_len);
        if (packret != 0) {
            uart0_sendStr("LAN UDP Send err!");
        }
        break;
        default:
        uart0_sendStr("Pack is not ssdq req!");
        break;
    }
}
/*
```

```
 * 硬件平台相关，创建UDP套接字，监听12476端口，与AirKiss库使用无关
 */
static void mm_startlandiscover(void){
    airkiss_udp.local_port = 12476;
    ptrairudpconn.type = ESPCONN_UDP;
    ptrairudpconn.proto.udp = &(airkiss_udp);
    espconn_create(&ptrairudpconn);
    espconn_regist_recvcb(&ptrairudpconn,wifilan_recv_callbk);
    os_timer_setfn(&time_serv, (os_timer_func_t *)time_callback,
    NULL);
    os_timer_arm(&time_serv, 5000, 1);//5s定时器
}
/*
 * 硬件平台初始化，与AirKiss库使用无关
 */
void  ICACHE_FLASH_ATTR
user_init(void)
{
    uart_init(115200, 115200);
    //这里假设设备已经连接上WiFi了，直接开始创建UDP
    system_init_done_cb(mm_startlandiscover);
}
```

下面对上面的示例代码进行分析：

文件开头定义并初始化了一些与内网发现功能相关的全局变量：

```
//用于发送上线通知包的定时器，平台相关
os_timer_t time_serv;
//UDP套接字相关变量，平台相关
esp_udp airkiss_udp;

struct espconn ptrairudpconn;

//用于缓存回包的数据缓冲区，也可以为局部变量
uint8_t lan_buf[200];

uint16_t lan_buf_len;

#define DEVICE_TYPE   "gh_27098xxxxx"

#define DEVICE_ID     "BD5D7899xxxxx"
#define DEFAULT_LAN_PORT    12476

//定义AirKiss库需要用到的一些标准函数，由对应的硬件平台提供，前三个为必要函数
const airkiss_config_t akconf =
{
    (airkiss_memset_fn)&memset,
    (airkiss_memcpy_fn)&memcpy,
    (airkiss_memcmp_fn)&memcmp,
    0
};
```

图 12 代码示例

ESP8266平台的入口函数为user_init(void) ,在该函数中只是初始了一下串口，连接 WiFi 的逻辑这里不做赘述，默认已经联网了，直接开始创建 UDP 接口，对 12476 端口进行监听，最后启动定时器用于定时发送设备在线消息，提高设备被微信客户端发现的速度：

```
/*
 * 硬件平台相关，创建UDP套接字，监听12476端口，与AirKiss库使用无关
 */
static void mm_startlandiscover(void){
    airkiss_udp.local_port = 12476;
    ptrairudpconn.type = ESPCONN_UDP;
    ptrairudpconn.proto.udp = &(airkiss_udp);
    espconn_create(&ptrairudpconn);
    espconn_regist_recvcb(&ptrairudpconn,wifilan_recv_callbk);
    os_timer_setfn(&time_serv, (os_timer_func_t *)time_callback, NULL);

    os_timer_arm(&time_serv, 5000, 1);//5s定时器
}
```

图 13 代码示例

在 UDP 的接收处理函数中，将收到的报文传递给 airkiss_lan_recv()函数，如果该函数返回 AIRKISS_LAN_SSDP_REQ，则调用函数 airkiss_lan_pack()完成响应报文的打包，最后通过 UDP 回复给发送方：

```c
/*
 * 硬件平台相关，UDP监听端口数据接收处理函数
 */

void wifilan_recv_callbk(void *arg, char *pdata, unsigned short len)
{
    airkiss_lan_ret_t ret = airkiss_lan_recv(pdata, len, &akconf);
    airkiss_lan_ret_t packret;
    switch (ret){
        case AIRKISS_LAN_SSDP_REQ:
        airkiss_udp.remote_port = DEFAULT_LAN_PORT;
        lan_buf_len = sizeof(lan_buf);
        packret = airkiss_lan_pack(AIRKISS_LAN_SSDP_RESP_CMD, DEVICE_TYPE, DEVICE_ID, 0, 0,
            lan_buf, &lan_buf_len, &akconf);
        if (packret != AIRKISS_LAN_PAKE_READY) {
            uart0_sendStr("Pack lan packet error!");
            return;
        }
        packret = espconn_sent(&ptrairudpconn, lan_buf, lan_buf_len);
        if (packret != 0) {
            uart0_sendStr("LAN UDP Send err!");
        }
        break;
        default:
        uart0_sendStr("Pack is not ssdq req!");
        break;
    }
}
```

图 14 代码示例

上面是通过接收微信客户端发送的服务发现请求并进行响应的方式，从而实现内网设备的发现功能。为了提高设备被微信客户端发现的速度，设备也可以定时向网络发送上线通知：

```c
/*
 * 平台相关定时器中断处理函数，比较正确的做法是在中断里面发送信号通知任务发送，这里
 * 为了方便说明直接发送
 */
static void time_callback(void)
{
    airkiss_udp.remote_port = DEFAULT_LAN_PORT;
    airkiss_udp.remote_ip[0] = 255;
    airkiss_udp.remote_ip[1] = 255;
    airkiss_udp.remote_ip[2] = 255;
    airkiss_udp.remote_ip[3] = 255;
    lan_buf_len = sizeof(lan_buf);
    ret = airkiss_lan_pack(AIRKISS_LAN_SSDP_NOTIFY_CMD, DEVICE_TYPE, DEVICE_ID, 0, 0,
        lan_buf, &lan_buf_len, &akconf);
    if (ret != AIRKISS_LAN_PAKE_READY) {
        uart0_sendStr("Pack lan packet error!");
        return;
    }
    ret = espconn_sent(&ptrairudpconn, lan_buf, lan_buf_len);
    if (ret != 0)
    {
        uart0_sendStr("UDP send error!");
    }
    uart0_sendStr("Finish send notify!");
}
```

图 15 代码示例

需要注意的是上线通知的目的 IP 为广播地址 255.255.255.255，目的端口为 12476，pack 的包类型为 AIRKISS_LAN_SSDP_NOTIFY_CMD。实现上述逻辑代码后，在同一个局域网内，进入微信客户端内网发现页面就可以发现设备了：

图 16 微信客户端内网发现示例

5. 注意事项

A．在 airkiss.h 文件中开启 AES 宏以后，注意链接文件要使用有带 AES 加密功能的静态库，屏蔽该宏以后要使用不带 AES 加密功能的静态库：

```
airkiss.h
1  /*
2   * airkiss.h
3   *
4   *  Created on: 2015-1-26
5   *      Author: peterfan
6   */
7
8  #ifndef AIRKISS_H_
9  #define AIRKISS_H_
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15 /*
16  * 定义AIRKISS_ENABLE_CRYPT为1以启用AirKiss加密功能
17  */
18 #ifndef AIRKISS_ENABLE_CRYPT
19 #define AIRKISS_ENABLE_CRYPT 1
20 #endif
```

图 17 代码示例

B．咨询 AirKiss 库相关问题时，请先调用 `airkiss_version()` 接口，并把该接口返回的内容直接打印出来：

```
/*
 * 获取AirKiss库版本信息
 */
const char* airkiss_version();
```

图 18 代码示例

6. 实验测试结果

已验证通过的路由器列表见下表(测试环境 15 个热点以上，多个存在中继器)，如果测试不通过先把模块靠近路由器再进行尝试排除信号问题导致的丢包。

序号	厂家	具体型号
1	华三（H3C）	WA2620-AGN
2	友讯(D-Link)	DIR-616
3	磊科(Netcore)	NW737
4	腾达(Tenda)	N6
5	极客科技(HiWiFi)	HC5661
6	思科(CISCO)	CVR100W
7	必联(LB-LINK)	BL-845R
8	腾达(Tenda)	N300 V2

表 1 测试实验路由器列表

7. Q&A

Q1：无法锁定 WiFi 信道

A1：先确认是否有正确处理 `airkiss_recv()` 函数的返回值，一旦该函数返回 `AIRKISS_STATUS_CHANNEL_LOCKED` 则需要立即停止切换信道，该类型返回值在整个 `AirKiss` 流程中只返回一次。如果已经正确处理还是无法锁定信道，请利用抓包工具在用一个环境下进行抓包，确认发送端是否有发包，并且路由器有进行转发（标志为 `FromDS=1 && ToDS=0`）；如果路由器没有进行转发或丢包严重可以尝试更换路由器试试，同时欢迎把路由器的型号反馈给我们：`http://bbs.iot.weixin.qq.com`，`22585` 之后的版本支持抓取手机包，请确认模块是否支持抓取手机包（标志为 `FromDS=0 && ToDS=1`）；

Q2：设备端 AES 解密失败

A2 :先确认不加密的情况下是否能成功接收 ,然后再确认通过 jsapi 传递的密钥或者 AirKiss Demo 测试软件上填写的密钥是否与设备内部的 AES 密钥一致，密钥为 128bit；

Q3：如何确定 AirKiss 流程完成

A3 :请先确保模块连接上路由器以后 ,通过 UDP 向目的端口 10000 发送至少 20 个广播包，包内容为一个字节的通过 `airkiss_get_result()` 获得的 `random` 值。如果是公众号通过 JSAPI 调起 AirKiss 功能的话，微信客户端在收到正确的 UDP 广播包后会弹窗提示并停止发送 AirKiss 包；如果是利用 AirKiss Demo 测试软件测试的话，在收到正确的 UDP 广播包以后，下方会弹出”Bingo”的提示并停止发包；

Q4：如何验证 AirKiss 静态库是否能够在某个平台正常使用？

A4 :如果编译和链接能够通过 ,并且调用 airkiss_version()函数能够返回正确的版本信息，说明 AirKiss 静态库适用于该平台。

Q5：airkiss_recv()参数和返回值的详细说明

A5：int airkiss_recv(airkiss_context_t* context, const void* frame, unsigned short length);

frame：

从混杂模式抓到的 802.11 网络帧，airkiss_recv 目前仅会访问 frame 所指向内存的前 24

字节，帧格式如下：

```
/*
 * Generic definitions for IEEE 802.11 frames.
 */
typedef struct
{
   unsigned char fc[2];  /* frame control */
   unsigned char dur[2]; /* duration/ID */
   unsigned char addr1[6];  /* address 1 */
   unsigned char addr2[6];  /* address 2 */
   unsigned char addr3[6];  /* address 3 */
   unsigned char seq[2]; /* sequence control */
} __ieee80211_frame;
```

（注意：有些平台在混杂模式下无法直接抓到 802.11 网络帧，请自行进行转换）

length：

802.11 网络帧的总长度（为避免出错，请直接传入混杂模式 API 返回的长度）

返回值：

22585 及之前的版本会返回如-429 和-669 之类的负值，这都是正常的，不会影响 AirKiss 解码过程，直接忽略即可。

Q6：为什么看不到 AirKiss 静态库打印出的 log？

A6：

1）请使用带 log 功能的静态库：libairkiss_log.a，libairkiss_aes_log.a；

2）请确保 airkiss_config_t 的 printf 函数有正确赋值。

3）请确保传入 airkiss_init()函数的 airkiss_config_t 为全局变量，如果调用 airkiss_recv()函数之后出现 crash，通常都是这个原因引起的；

4）目前只有 airkiss_recv()函数会打印 log，其它函数都不会产生 log。

Q7：正常情况下 AirKiss 解码全过程的 log 是什么样的？

A7：

```
----------  airkiss_version() "airkiss-1.0.0-22585(May 11 2015
17:10:22);iccarm/V7.40.2.8542;Cortex-M3"
```

```
---------- airkiss_init() ok
AirKiss recv guide field:  0, mac_crc 0x3c, len  64, seq_count 0,
recv_count 0
AirKiss recv guide field:  1, mac_crc 0xd4, len  64, seq_count 0,
recv_count 0
AirKiss recv guide field:  1, mac_crc 0xd4, len  64, seq_count 0,
recv_count 1
AirKiss recv guide field:  1, mac_crc 0xd4, len  64, seq_count 0,
recv_count 2
AirKiss recv guide field:  2, mac_crc 0xc5, len  64, seq_count 0,
recv_count 0
AirKiss recv guide field:  1, mac_crc 0xd4, len  64, seq_count 0,
recv_count 3
AirKiss recv guide field:  2, mac_crc 0xc5, len  64, seq_count 0,
recv_count 1
AirKiss recv guide field:  3, mac_crc 0xfa, len  64, seq_count 0,
recv_count 0
AirKiss recv guide field:  3, mac_crc 0xfa, len  64, seq_count 0,
recv_count 1
AirKiss recv guide field:  4, mac_crc 0x93, len  64, seq_count 0,
recv_count 0
AirKiss recv guide field:  4, mac_crc 0x93, len  84, seq_count 0,
recv_count 1
AirKiss recv guide field:  4, mac_crc 0x93, len  84, seq_count 0,
recv_count 2
AirKiss recv guide field:  5, mac_crc 0x23, len  64, seq_count 0,
recv_count 0
AirKiss recv guide field:  5, mac_crc 0x23, len  64, seq_count 0,
recv_count 1
AirKiss recv guide field:  5, mac_crc 0x23, len  64, seq_count 0,
recv_count 2
AirKiss recv guide field:  5, mac_crc 0x23, len  64, seq_count 0,
recv_count 3
```

```
AirKiss recv guide field:  6, mac_crc 0x29, len  98, seq_count 0,
recv_count 0
AirKiss recv guide field:  6, mac_crc 0x29, len 118, seq_count 0,
recv_count 1
AirKiss recv guide field:  7, mac_crc 0x2d, len  64, seq_count 0,
recv_count 0
AirKiss recv guide field:  7, mac_crc 0x2d, len  84, seq_count 0,
recv_count 1
AirKiss recv guide field:  8, mac_crc 0x4e, len  84, seq_count 0,
recv_count 0
AirKiss recv guide field:  9, mac_crc 0x10, len  64, seq_count 0,
recv_count 0
AirKiss recv guide field: 10, mac_crc 0xe5, len  81, seq_count 0,
recv_count 0
AirKiss recv guide field: 10, mac_crc 0xe5, len  83, seq_count 0,
recv_count 1
AirKiss recv guide field: 10, mac_crc 0xe5, len  84, seq_count 1,
recv_count 2
AirKiss recv guide field: 10, mac_crc 0xe5, len  81, seq_count 0,
recv_count 3
AirKiss recv guide field: 10, mac_crc 0xe5, len  82, seq_count 1,
recv_count 4
AirKiss recv guide field: 10, mac_crc 0xe5, len  83, seq_count 2,
recv_count 5
AirKiss recv guide field: 10, mac_crc 0xe5, len  84, seq_count 3,
recv_count 6
AirKiss recv guide field: length offset 80, mac_s c0:ee:fb:03:44:0c,
mac_a f8:e9:03:b5:40:da
---------- airkiss_recv() channel locked
AirKiss recv magic field: data length 31, crc 0x0c
AirKiss recv prefix field: pwd length 16, crc 0x9d
AirKiss recv data: index  0, crc 0x1a, length 4/31
AirKiss recv data: index  1, crc 0x2f, length 8/31
```

```
AirKiss recv data: index  2, crc 0x2c, length 12/31
AirKiss recv data: index  4, crc 0x3a, length 16/31
AirKiss recv data: index  5, crc 0x63, length 20/31
AirKiss recv data: index  6, crc 0x31, length 24/31
AirKiss recv data: index  7, crc 0x03, length 27/31
AirKiss recv data: index  3, crc 0x41, length 31/31
AirKiss decrypt pwd: length 16, pwd CF 6F 2A 60 CD F9 85 F4 B2 23
02 57 F9 A5 4F 92
AirKiss complete: ssid "D-Link_DIR_616", pwd "TTkupao916", random
0x71
----------  airkiss_get_result() ok
ssid = "D-Link_DIR_616", pwd = "TTkupao916", ssid_length = 14,
pwd_length = 10, random = 0x71
```

Q8：在混杂模式下抓到的 802.11 网络帧，传递给 airkiss_recv()函数之前需要做过滤吗？

A8：不需要做任何过滤和预处理，收到的任何包都可以直接传入 airkiss_recv()函数，该函数内部对传入的帧已经按各种条件进行了过滤。

Q9：airkiss_recv()函数返回了 AIRKISS_STATUS_CHANNEL_LOCKED，但是锁定了错误的 channel，这种情况正常吗？应该怎么处理？

A9：这种情况是正常的，假设要配置的 WiFi 接入点在 channel 6，因为在 channel 7 和 channel 8 也能抓到包，AirKiss 有可能会锁定到 channel 7 或 channel 8 上，因为这两个 channel 信号相对较弱，导致 AirKiss 后续解码过程失败。

建议处理方案：

1)在启动 AirKiss 之前进行信道预扫描 ,比如扫描到 3 个 ssid ,对应的 channel 分别是“1，6，11”，那么后续切换 channel 的时候就只用在这 3 个 channel 切换即可；

2) 22585 之后的版本对 AirKiss 算法做了优化，大大减少了锁定错误 channel 的几率；

3) 22585 之后的版本还提供一个清缓存的接口，用户在切换信道以后可以调用一下清除在上一个信道接收到的数据，也可以在一定程度上减少锁定错误的几率；

4) 另外跟厂家模块确认好模块的接收灵敏度，据了解有些模块天线部分处理不好，会接收到相隔几个信道上的数据。

Q10：使用不加密的 AP，AirKiss 的密码设置为空，并且启用 AES 加密的情况下 AirKiss 会失败是什么原因？

A10：22585 之前的版本有这个 bug，开启 log 可以看到如下信息：

```
AirKiss decrypt failed: error XXX
```

请升级 AirKiss 静态库到 22585 或更新的版本。

Q11 ： airkiss_recv() 一直返回 AIRKISS_STATUS_CONTINUE ，无法返回 AIRKISS_STATUS_CHANNEL_LOCKED 或者 AIRKISS_STATUS_COMPLETE，应该怎么处理？

A11：

1）22585 及之前的版本只会抓 AP 的包（即 FromDS=1 && ToDS = 0），而某些路由器对广播包有一定的策略限制，导致设备抓不到广播包。开发的时候尽量多找几种路由器来测试，排除路由器的因素；

2）22585 之后的版本将会支持同时抓手机的包（即 FromDS=0 && ToDS = 1），可以大大提高 AirKiss 的兼容性和成功率。

Q12：AirKiss 静态库的 AES 加密功能有什么作用？

A12：使用 AirKiss 技术给设备配置 WiFi 的过程中，AirKiss 发送方（微信客户端或者 AirKiss 测试工具）会以特定的方式将 WiFi 的 ssid 和密码广播出来，为保证 WiFi 密码不被恶意拦截，在 AirKiss 配置过程中需要对 WiFi 密码进行加密，设备端接收到以后再进行解密。

Q13：AirKiss 静态库是否支持使用自定义的 AES 解密算法？

A13：支持。为节省代码空间，如果您的平台自带 AES 解密能力，请使用不支持 AES 的 AirKiss 库，AirKiss 配置成功后再自行对 WiFi 密码进行解密。AirKiss 使用的 AES 参数如下：

AES 128，CBC，PKCS7Padding（注意：ssid 始终是明文的，只用对 WiFi 密码进行解密）

Q14：使用支持 AES 的 AirKiss 静态库是否可以不使用加密功能？

A13：可以。为方便起见，使用支持 AES 的 AirKiss 静态库（libairkiss_aes.a 或 libairkiss_aes_log.a），如果不想使用加密功能，不调用 `airkiss_set_key()` 即可。