



# AirKiss 库文件使用指南

WIFI 芯片/模组如何实现 AirKiss

发布时间：2015-04-02

版本：1.1

---

## 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

## 评审记录

评审人	评审时间	评审内容	评审意见

## 目标读者

WIFI 芯片开发人员，WIFI 模块调试人员

---

## 目录

<b>1</b>	<b>什么是 AIRKISS™技术.....</b>	<b>1</b>
1.1	AIRKISS 技术使用场景.....	1
1.2	AIRKISS 技术应用实例.....	1
1.3	AIRKISS 技术的优势.....	4
<b>2.</b>	<b>AIRKISS 库组成结构介绍 .....</b>	<b>4</b>
2.1	AIRKISS 库文件组成.....	4
<b>3.</b>	<b>AIRKISS 库使用流程说明 .....</b>	<b>5</b>
3.1	设备平台能力要求.....	5
3.2	AIRKISS 库使用说明.....	6
3.3	AIRKISS 库调用流程图.....	7
<b>4.</b>	<b>AIRKISS 库使用示例 .....</b>	<b>8</b>
<b>5.</b>	<b>注意事项.....</b>	<b>15</b>
<b>6.</b>	<b>实验测试结果 .....</b>	<b>16</b>
<b>7.</b>	<b>Q&amp;A .....</b>	<b>16</b>

# 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。

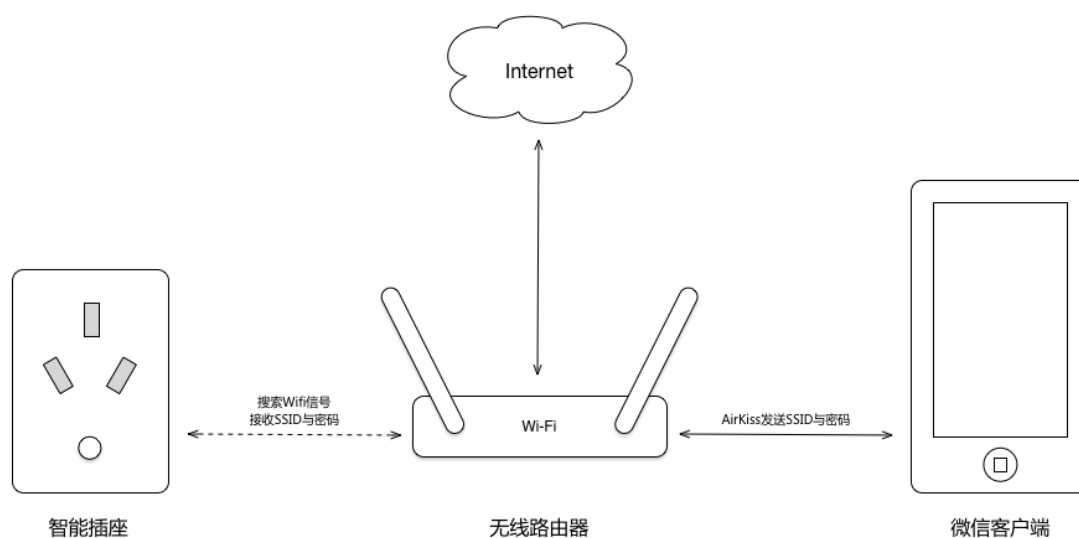


图 1 AirKiss 配置设备联网示意图

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 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 128 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 akcontext;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(&akcontext, &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(&akcontext, key, strlen(key));

```

```
/*
 * 定义AIRKISS_ENABLE_CRYPT为1以启用AirKiss加密功能
 */
#ifdef 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：

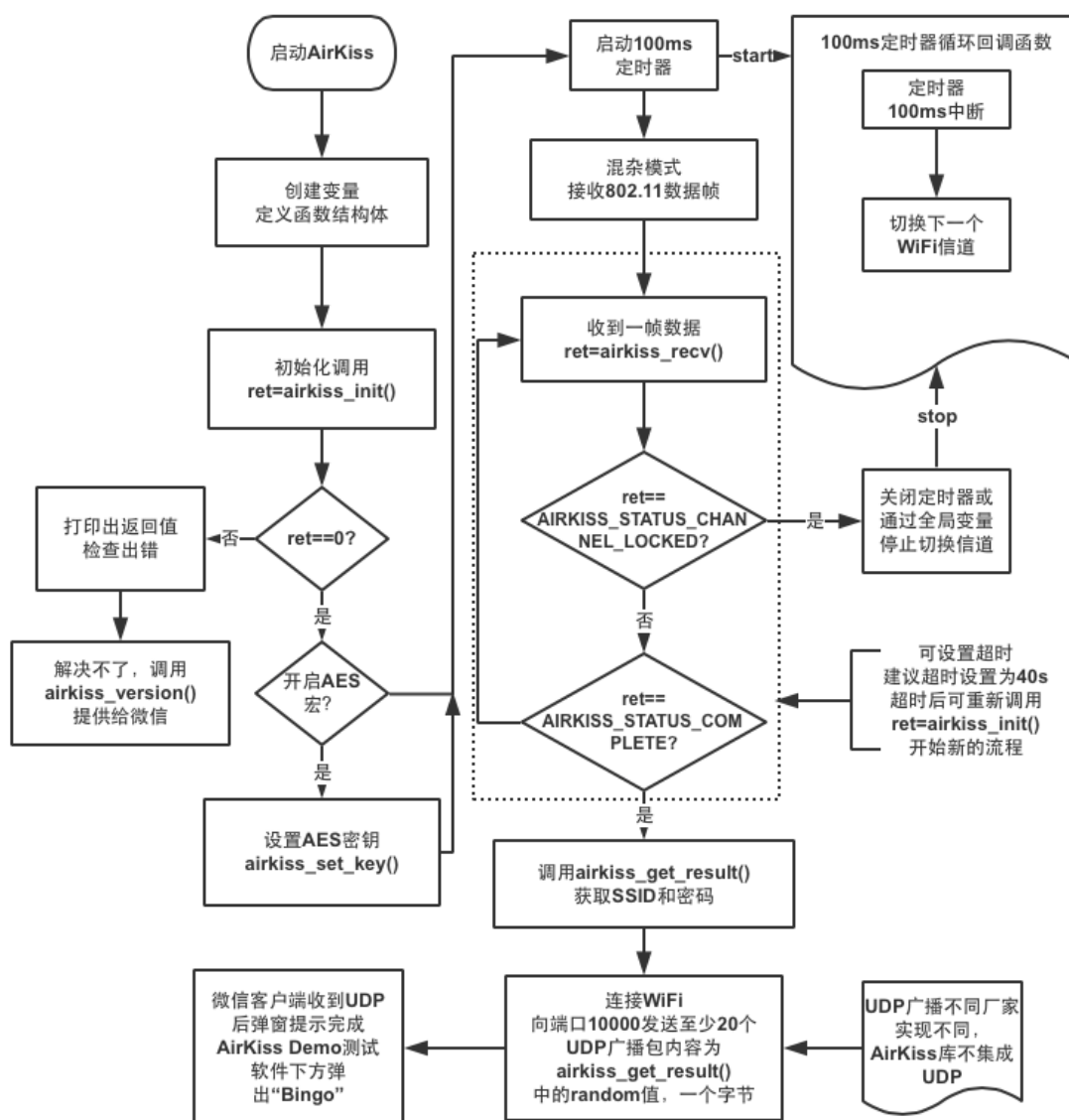


图 4 AirKiss 工作流程示意图

## 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 akcontext;

//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 *akcontextprt;
//akcontextprt =
//(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(&akcontext, 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 = "Wechatiohardwav";

    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(&akcontext, &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(&akcontext, 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 akcontext;
//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 *akcontextprt;
//示例: akcontextprt = (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 5 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(&akcontext, &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(&akcontext, 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 6 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：

```
30 /*
31  * 平台相关定时器中断处理函数，100ms中断后切换信道
32  */
33 static void time_callback(void)
34 {
35     //切换信道
36     if (cur_channel >= 13) cur_channel = 1;
37     else cur_channel++;
38     wifi_set_channel(cur_channel);
39 }
40
```

图 7 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(&akcontext, 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 8 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：

```
1 /*
2  * airkiss成功后读取配置信息，平台无关，修改打印函数即可
3  */
4 static void airkiss_finish(void){
5     int8_t err;
6     uint8 buffer[256];
7     airkiss_result_t result;
8     err = airkiss_get_result(&akcontext, &result);
9     if (err == 0)
10     {
11         uart0_sendStr("----- airkiss_get_result() ok!");
12         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);
13         uart0_sendStr(buffer);
14     }
15     else
16     {
17         uart0_sendStr("----- airkiss_get_result() failed !\r\n");
18     }
19 }
```

图 9 代码示例

示例代码调试输出数据内容如下所示：

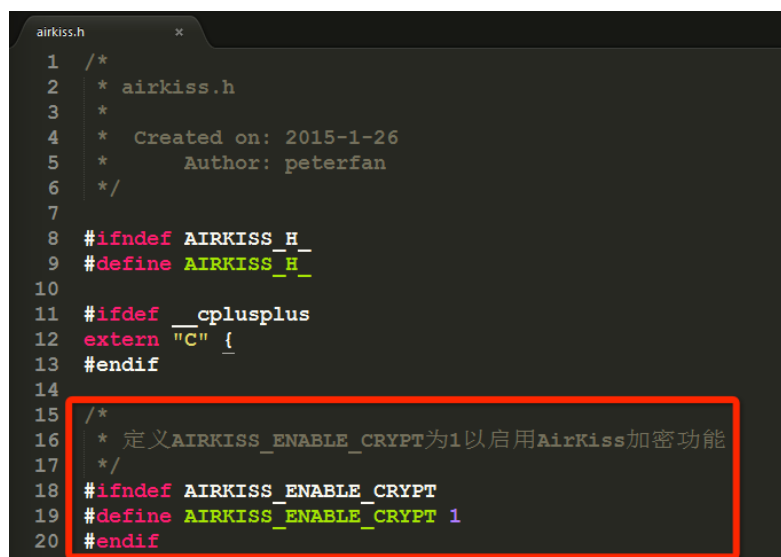
```
Start airkiss!
Finish init airkiss!
----- airkiss_get_result() ok!ssid = "Tencent-GuestWiFi", pwd =
" ", ssid_length = 17, pwd_length = 0, random = 0x04
```

图 10 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。

## 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：



```

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
21

```

图 11 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 12 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

Q：无法锁定 WiFi 信道

A：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，请利用抓包工具在同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>；

Q：设备端 AES 解密失败

A：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；

Q：如何确定 AirKiss 流程完成

A：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。