



Hi3861V100 / Hi3861LV100 ANY 软件

## 开发指南

文档版本 01

发布日期 2020-04-30

版权所有 © 上海海思技术有限公司2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 商标声明



**HISILICON**、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## 注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

# 上海海思技术有限公司

地址：            深圳市龙岗区坂田华为总部办公楼    邮编：518129

网址：            <https://www.hisilicon.com/cn/>

客户服务邮箱：  [support@hisilicon.com](mailto:support@hisilicon.com)



# 前言

## 概述

本文档详细介绍了Hi3861V100、Hi3861LV100 WiFi软件ANY功能的接口以及开发流程。

## 产品版本

与本文档对应的产品版本如下。

产品名称	产品版本
Hi3861	V100
Hi3861L	V100



## 读者对象

本文档主要适用以下工程师：



- 技术支持工程
- 软件开发工程师

## 符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 <b>危险</b>	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 <b>警告</b>	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。



符号	说明
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

## 修改记录

文档版本	发布日期	修改说明
01	2020-04-30	第一次正式版本发布。
00B01	2020-04-03	第一次临时版本发布。



## 目录

前言.....	i
1 概述.....	1
2 ANY 功能初始化和去初始化.....	2
2.1 概述.....	2
2.2 开发流程.....	2
2.3 注意事项.....	3
2.4 编程实例.....	3
3 ANY 设备扫描和发现.....	4
3.1 概述.....	4
3.2 开发流程.....	4
3.3 注意事项.....	5
3.4 编程实例.....	5
4 ANY 通信.....	6
4.1 概述.....	6
4.2 开发流程.....	6
4.3 注意事项.....	8
4.4 编程实例.....	8
5 ANY API 使用示例.....	9
5.1 概述.....	9
5.2 代码实现.....	10
5.3 运行结果.....	16
6 ANY 设备配对 Demo.....	17
6.1 概述.....	17
6.2 配对 Demo 实现方案.....	17
6.3 注意事项.....	18



# 1 概述

ANY功能是一种华为私有的短数据通信功能，允许处于同一信道的2个Wi-Fi设备进行直接的点对点无连接通信。Hi3861V100、Hi3861LV100 提供了相关的ANY API（Application Programming Interface）供应用层使用，以实现ANY应用。ANY API用于ANY功能的初始化和去初始化、扫描发现ANY设备，以及对端ANY设备进行通信。ANY可以应用于智能开关控制灯泡、传感器数据采集、遥控器控制家用电器等无线控制场景。

ANY功能特点如下：

- 每个设备可以选择一个接口（例如：wlan0或ap0）用于ANY报文的收发。
- ANY报文采用接口当前所在信道进行收发，和通信对端需要处于同一信道。
- 单个ANY报文最多可以支持250byte的用户层数据。
- 单个ANY设备最多支持同时和16个ANY对端设备进行通信，其中最多允许和6个对端进行加密通信。
- ANY设备可以收发ANY单播报文和ANY广播报文，不支持组播报文。
- ANY设备可以扫描发现附近的其他ANY设备。

## 说明

API详细说明请参见《Hi3861V100/Hi3861LV100 API 开发参考》。



# 2 ANY 功能初始化和去初始化

## 2.1 概述

## 2.2 开发流程

## 2.3 注意事项

## 2.4 编程实例

## 2.1 概述

ANY初始化是指选择一个已经创建完成的STA或SoftAP作为本设备ANY收发通信的接口，在该接口上使能ANY功能。经过ANY初始化之后，ANY采用被选择的接口所在的信道进行收发通信，并采用该接口的MAC地址作为发送源地址和接收目的地址。其中：

- 对于SoftAP，所在信道为创建时指定的信道。
- 对于STA，如果关联了某一SoftAP，所在信道为该SoftAP指定的信道；如果未关联，则可以采用设置信道的API函数hi\_wifi\_set\_channel给STA指定一个信道。

ANY去初始化是指去使能ANY功能，去初始化操作会清除所有ANY相关的配置信息，包括ANY对端设备的MAC地址和通信密钥。如果已经初始化ANY功能，需要执行去初始化之后才能重新初始化。

## 2.2 开发流程

### 使用场景

当需要使能或去使能ANY功能时使用。

### 功能

ANY功能初始化和去初始化提供的接口如[表2-1](#)所示。



表 2-1 ANY 初始化和去初始化接口描述

接口名称	描述
hi_wifi_any_init	选择一个接口初始化ANY功能，参数为接口名称。
hi_wifi_any_deinit	去初始化ANY功能。

## 开发流程

ANY应用初始化和去初始化的典型流程：

- 步骤1 调用hi\_wifi\_sta\_start，启动STA；或调用hi\_wifi\_softap\_start，启动SoftAP。
- 步骤2 调用hi\_wifi\_any\_init，选择STA接口（ wlan0 ）或AP接口（ ap0 ）作为ANY通信接口。
- 步骤3 根据业务具体需求，进行ANY扫描发现或ANY通信。
- 步骤4 调用hi\_wifi\_any\_deinit，去初始化ANY功能。

----结束

## 返回值

ANY初始化和去初始化接口的返回值如表2-2所示。

表 2-2 ANY 初始化和去初始化返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HI_FAIL	1	执行失败。

## 2.3 注意事项

- 一个设备只允许选择一个接口作为ANY收发接口（即进行一次初始化），重新初始化之前需要先调用ANY去初始化。
- ANY去初始化之后，所有ANY相关的配置信息会被清除。
- 在STA或SoftAP接口上初始化ANY之后，关闭STA或关闭SoftAP过程会自动去初始化ANY功能。

## 2.4 编程实例

ANY在STA接口上初始化的示例请参见“[5 ANY API使用示例](#)”中 example\_any\_with\_sta函数实现。

ANY在SoftAP接口上初始化的示例请参见“[5 ANY API使用示例](#)”中 example\_any\_with\_ap函数实现。





# 3 ANY 设备扫描和发现

- 3.1 概述
- 3.2 开发流程
- 3.3 注意事项
- 3.4 编程实例

## 3.1 概述

初始化ANY之后，可以通过扫描发现API发现附近支持ANY的设备，获取设备MAC地址、所在信道、RSSI强度等信息。

## 3.2 开发流程

### 使用场景

ANY初始化之后，扫描发现周围ANY设备。

### 功能

ANY设备扫描和发现提供的接口如[表3-1](#)所示。

表 3-1 ANY 设备扫描和发现接口描述

接口名称	描述
hi_wifi_any_discover_peer	注册扫描完成回调函数并执行一次ANY扫描。 调用该函数之前用户需要实现 hi_wifi_any_scan_result_cb类型的扫描完成回调函数，对于该回调函数，驱动传入的输入参数为发现的ANY设备的指针数组和数组元素个数，每个元素指向一个发现的ANY设备相关信息。



## 开发流程

在ANY功能初始化之后才能进行设备发现，开发流程如下：

**步骤1** 用户需要实现hi\_wifi\_any\_scan\_result\_cb回调函数，用于处理ANY扫描完成之后上报的结果。

**步骤2** 调用hi\_wifi\_any\_discover\_peer，向驱动注册上述回调函数并启动一次ANY扫描。

----结束

## 返回值

ANY发起扫描接口的返回值如表3-2所示。

表 3-2 ANY 发起扫描返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HI_FAIL	1	执行失败。

## 3.3 注意事项

- 无论是STA还是SoftAP，只要初始化了ANY，就能执行ANY扫描、发现周围其他ANY设备，同时也能被其他ANY设备扫描到。
- 扫描结果中，如果发现的ANY设备是SoftAP，则设备信息中会包含该SSID信息；如果是STA，则不包含SSID（为空字符串）。
- 回调函数运行于驱动线程，不能阻塞或长时间等待，建议新建一个任务处理扫描结果，回调函数向该任务复制结果之后则退出运行。
- 单次扫描最多通过回调函数返回32个对端设备信息，回调函数传入的数组内存由驱动自行管理，在回调函数中不应释放。

## 3.4 编程实例

发起ANY扫描的示例请参见“[5 ANY API使用示例](#)”中example\_any\_with\_sta函数实现，示例中没有找到目标设备则持续进行扫描。

用户编写扫描完成回调函数的示例请参见“[5 ANY API使用示例](#)”中wifi\_any\_scan\_result\_cb的实现，示例中匹配到目标设备则通知消息处理任务向该设备发起ANY通信。用户实现的扫描完成回调函数中也可以申请内存，复制所有扫描结果发给消息处理任务处理，然后快速退出。



# 4 ANY 通信

- 4.1 概述
- 4.2 开发流程
- 4.3 注意事项
- 4.4 编程实例

## 4.1 概述

ANY设备支持和同信道的其他ANY设备进行加密或不加密通信，单个ANY报文最多可以支持250byte的用户数据。如果进行加密通信，要求通信双方提前配置同样的16byte长度密钥。用户需要实现接收回调函数来处理接收到的对端消息，同时需要实现发送完成回调函数，驱动向对端发送ANY消息之后，通过调用该回调函数反馈是否最终发送成功（是否收到对端的ACK报文）。

## 4.2 开发流程

### 使用场景

ANY初始化之后，和处于同信道的对端ANY设备通信。

### 功能

ANY通信提供的接口如[表4-1](#)所示。



表 4-1 ANY 通信接口描述

接口名称	描述
hi_wifi_any_set_callback	通信之前注册ANY报文的接收回调函数和发送完成回调函数。 用户需要实现hi_wifi_any_recv_cb类型的接收回调函数，注册该回调函数之后，驱动收到ANY报文则调用该函数传递给应用层；用户可以实现hi_wifi_any_send_complete_cb类型的发送完成回调函数，每完成一次发送，驱动调用该函数将发送结果（是否发送成功并收到ACK）反馈给应用层，发送成功则status为1，失败则为其他值，如果不需要该函数，则注册为NULL。
hi_wifi_any_add_peer	添加ANY对端设备信息，主要包括MAC地址、用于和对端加密通信的密钥，对端设备需配置同样密钥才能解析加密数据。
hi_wifi_any_del_peer	删除指定MAC地址的对端设备信息。
hi_wifi_any_fetch_peer	获取指定索引的对端设备信息，一般用于遍历查询所有已配置的对端设备。
hi_wifi_any_send	向指定MAC地址的对端设备发送ANY数据，每次发送不超过250byte。

## 开发流程

在ANY功能初始化之后才能进行ANY通信，通信前需要确保和通信对端处于同一信道，开发流程如下：

- 步骤1** 实现ANY接收回调函数hi\_wifi\_any\_recv\_cb和发送完成回调函数hi\_wifi\_any\_send\_complete\_cb。
- 步骤2** 调用hi\_wifi\_any\_set\_callback，向驱动注册上述回调函数。
- 步骤3** 调用hi\_wifi\_any\_add\_peer，配置密钥，使之绑定对端设备的MAC地址（如果不配置密钥，则进行明文通信）。
- 步骤4** 调用hi\_wifi\_any\_send，向对端设备发送数据，如果对端设备的MAC地址已绑定密钥，则会采用该密钥加密通信数据。

----结束

## 返回值

ANY配置对端信息和发送接口的返回值如表4-2所示。

表 4-2 ANY 通信函数返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。



序号	定义	实际数值	描述
2	HI_FAIL	1	执行失败。

### 4.3 注意事项

- 接收和发送完成回调函数运行于驱动线程，不能阻塞或长时间等待，建议新建一个任务处理收发报文，回调函数中向新任务复制数据之后退出。
- 一个ANY设备可以支持和16个对端通信，其中最多和6个对端加密通信，通信双方需要配置相同密钥。
- 驱动传给接收回调函数的数据内存由驱动自行管理，接收回调函数中不应释放。
- ANY只在接口所在的当前信道收发数据，如果对端在其他信道，需要先调用切信道API切换信道之后再通信。

### 4.4 编程实例

2个ANY设备互相通信的示例请参见“[5 ANY API使用示例](#)”。



# 5 ANY API 使用示例

## 5.1 概述

## 5.2 代码实现

## 5.3 运行结果

## 5.1 概述

本示例实现了2个ANY设备之间点对点通信（一端为ANY STA，另一端为ANY SoftAP）：

- ANY STA在入口函数example\_any\_with\_sta中创建了一个STA，在该STA接口上初始化ANY功能；同时新建了一个消息处理任务，在该任务中进行ANY消息的收发和扫描结果的处理；最后执行ANY扫描搜索目标ANY设备。
- ANY SoftAP在入口函数example\_any\_with\_ap中创建了一个SoftAP并启动了DHCP服务器，之后在SoftAP接口上初始化了ANY功能并新建了一个消息处理任务进行ANY消息的收发。

### 须知

在接收回调函数wifi\_any\_recv\_cb和发送完成回调函数wifi\_any\_send\_cb的实现中，为了避免长时间阻塞驱动线程，执行简单复制将消息发给消息处理任务之后则退出。编译运行时，example\_any\_with\_sta可以挂接在STA设备的app\_main中，example\_any\_with\_ap挂接在SoftAP设备的app\_main中。

### 说明

本示例为明文通信，两端也可以在通信之前通过hi\_wifi\_any\_add\_peer接口给对端配置同样的16byte长度通信密钥，这样则可以进行加密通信。



## 5.2 代码实现

### 说明

本示例中，ANY STA和ANY SoftAP可以共享所有代码。对于ANY STA，在app\_main中调用example\_any\_with\_sta入口函数；对于ANY SoftAP，则调用example\_any\_with\_ap入口函数。

ANY初始化、扫描、通信的代码示例如下：

```
#include "lwip/netifapi.h"
#include "hi_wifi_api.h"
#include "hi_any_api.h"
#include "hi_types.h"
#include "hi_msg.h"
#include "hi_task.h"
#include "hi_mem.h"

#define ANY_TASK_PRIORITY 24
#define ANY_TASK_STACK_SIZE 2048
#define ANY_TASK_NAME "any_msg_task"
#define ANY_TASK_SLEEP_TIME 300
#define ANY_MSG_QUEUE_MAX_LEN 16
#define WIFI_IFNAME_MAX_SIZE 16
#define ANY_MSG_MAX_SIZE 50

static hi_u32 g_any_msg_queue = 0;
static hi_u32 g_any_msg_task_id = 0;
static hi_u8 g_any_msg[ANY_MSG_MAX_SIZE] = {0};
static hi_u8 g_find_any_device = 0;
static hi_s16 g_any_msg_seqnum = -1;
static hi_u8 g_task_running = 0;

typedef enum any_callback_enum {
    ANY_SCAN_CALLBACK,
    ANY_RECV_CALLBACK,
    ANY_SEND_COMPLETE_CALLBACK,
    ANY_CALLBACK_BUTT
}any_callback_msg_enum;

typedef struct {
    hi_u32 type;
    hi_u8 mac[WIFI_ANY_MAC_LEN];
    hi_u8 status;
    hi_u8 seqnum;
    hi_u8 *data;
    hi_u16 len;
    hi_u8 channel;
    hi_u8 resv;
}any_msg_stru;

void wifi_any_recv_proc(const unsigned char *mac, unsigned char *data, unsigned short len,
unsigned char seqnum)
{
    hi_u8 max_msg_num = 10;
    /* 收到同样序列号的帧，默认为是应用层重传帧，直接返回，由外面调用者释放内存 */
    if (g_any_msg_seqnum == seqnum) {
        return;
    }
    g_any_msg_seqnum = seqnum;
```



```
printf("{recv from MAC:0x%02x, seq:%d, len:%d, data:", mac[5], seqnum, len); /* 08:格式, 5:下标 */
for (unsigned short loop = 0; loop < len; loop++) {
    printf("%c", data[loop]);
}
printf(" }\r\n");

/* 向对端回复消息, 作为Sample, 回复有限条消息 */
if (seqnum < max_msg_num) {
    hi_wifi_any_send(mac, WIFI_ANY_MAC_LEN, g_any_msg, strlen((char *)g_any_msg), +seqnum);
}
return;
}

void wifi_any_send_complete_proc(const unsigned char *mac, unsigned char status, unsigned char seqnum)
{
    printf("{send to MAC:0x%02x, status: %d seq: %d}\r\n", mac[WIFI_ANY_MAC_LEN - 1], status, seqnum);
    return;
}

void wifi_any_scan_success_proc(const unsigned char *mac, unsigned char channel)
{
    /* 获取当前设备所在信道, 作为例子, ANY扫描从STA发起为接口名称为wlan0, 从AP发起则为ap0 */
    hi_u8 uc_len = strlen("wlan0") + 1;
    hi_s32 current_channel = hi_wifi_get_channel("wlan0", uc_len);
    /* 切换到其他信道发送ANY数据 */
    /* 注意: 如果在当前信道已有业务(如已关联某路由器), 切到其他信道会引起该业务中断 */
    if (((hi_u8)current_channel) != channel) {
        hi_wifi_set_channel("wlan0", uc_len, channel);
        printf("current channel %d switch to channel %d\r\n", current_channel, channel);
    }

    /* 向扫描到的目标ANY设备发送数据, 该帧设置发送序列号为1 */
    hi_wifi_any_send(mac, WIFI_ANY_MAC_LEN, g_any_msg, strlen((char *)g_any_msg), 1);
    return;
}

static hi_void* handle_any_msg(hi_void* data)
{
    (hi_void)data;
    int ret;
    any_msg_stru msg = {0};

    if (g_any_msg_queue == 0) {
        ret = hi_msg_queue_create(&g_any_msg_queue, ANY_MSG_QUEUE_MAX_LEN, sizeof(any_msg_stru));
        if (ret != HI_ERR_SUCCESS) {
            printf("create any message queue ret:%d\r\n", ret);
            g_any_msg_task_id = 0;
            return HI_NULL;
        }
    }

    while (1) {
        hi_u32 msg_size = sizeof(any_msg_stru);
        ret = hi_msg_queue_wait(g_any_msg_queue, (hi_pvoid)&msg, ANY_TASK_SLEEP_TIME, &msg_size);
        if (ret == HI_ERR_SUCCESS) {
```





```
switch (msg.type) {
    case ANY_SCAN_CALLBACK:
        wifi_any_scan_success_proc(msg.mac, msg.channel);
        break;
    case ANY_RECV_CALLBACK:
        wifi_any_recv_proc(msg.mac, msg.data, msg.len, msg.seqnum);
        hi_free(HI_MOD_ID_WIFI_DRV, msg.data);
        break;
    case ANY_SEND_COMPLETE_CALLBACK:
        wifi_any_send_complete_proc(msg.mac, msg.status, msg.seqnum);
        break;
    default:
        break;
}
} else {
    if (g_task_running == 0) {
        break;
    }
}
}
}
g_any_msg_task_id = 0;

hi_u8 trycount = 3;
while (trycount > 0) {
    if (hi_msg_queue_delete(g_any_msg_queue) == HI_ERR_SUCCESS) {
        g_any_msg_queue = 0;
        return HI_NULL;
    }
    trycount--;
}
printf("delete any msg queue failed!\r\n");
return HI_NULL;
}

hi_u32 write_any_msg(any_msg_stru *msg)
{
    hi_u32 ret;
    if ((g_any_msg_queue == 0) || (g_task_running == 0)) {
        printf("msg queue or task is not working!\r\n");
        return HI_ERR_FAILURE;
    }
    ret = hi_msg_queue_send(g_any_msg_queue, msg, 0, sizeof(any_msg_stru));
    return ret;
}

hi_u32 any_start_callback_task(hi_void)
{
    hi_u32 ret;
    if (g_any_msg_task_id != 0) {
        return HI_ERR_FAILURE;
    }

    hi_task_attr task_init = {0};
    task_init.task_prio = ANY_TASK_PRIORITY;
    task_init.stack_size = ANY_TASK_STACK_SIZE;
    task_init.task_name = ANY_TASK_NAME;
    g_task_running = 1;
    ret = hi_task_create(&g_any_msg_task_id, &task_init, handle_any_msg, HI_NULL);
    if (ret != HI_ERR_SUCCESS) {
        printf("create any msg task ret:%d\r\n", ret);
        return ret;
    }
}
```



```
    printf("create any msg task success!\r\n");
    return ret;
}

hi_u32 any_destory_callback_task(hi_void)
{
    printf("destory any callback task\r\n");
    g_task_running = 0;
    return HI_ERR_FAILURE;
}

void wifi_any_rcv_cb(unsigned char *mac, unsigned char *data, unsigned short len, unsigned
char seqnum)
{
    any_msg_stru msg = {0};

    msg.type = ANY_RECV_CALLBACK;
    if (memcpy_s(msg.mac, sizeof(msg.mac), mac, sizeof(msg.mac)) != EOK) {
        return;
    }
    msg.len = len;
    msg.seqnum = seqnum;
    msg.data = (unsigned char *)hi_malloc(HI_MOD_ID_WIFI_DRV, len);
    if (msg.data == NULL) {
        return;
    }
    if (memcpy_s(msg.data, len, data, len) != EOK) {
        hi_free(HI_MOD_ID_WIFI_DRV, msg.data);
        return;
    }
    if (write_any_msg(&msg) != HI_ERR_SUCCESS) {
        hi_free(HI_MOD_ID_WIFI_DRV, msg.data);
    }
    return;
}

void wifi_any_send_cb(unsigned char *mac, unsigned char status, unsigned char seqnum)
{
    any_msg_stru msg = {0};

    if (memcpy_s(msg.mac, sizeof(msg.mac), mac, sizeof(msg.mac)) != EOK) {
        return;
    }
    msg.type = ANY_SEND_COMPLETE_CALLBACK;
    msg.status = status;
    msg.seqnum = seqnum;
    if (write_any_msg(&msg) != HI_ERR_SUCCESS) {
        printf("write send complete failed\r\n");
    }
    return;
}

void wifi_any_scan_result_cb(hi_wifi_any_device *devices[], unsigned char num)
{
    unsigned char    target_ssid[] = "my_wifi";
    any_msg_stru     msg = {0};
    unsigned char    loop;

    if ((devices == NULL) || (num == 0)) {
        printf("Total scanned ANY dev num: 0\r\n");
        return;
    }
}
```



```
for (loop = 0; (loop < num) && (devices[loop] != NULL); loop++) {
    if (strcmp((char *)devices[loop]->ssid, (char *)target_ssid) != 0) {
        continue;
    }
    g_find_any_device = 1;
    msg.type = ANY_SCAN_CALLBACK;
    msg.channel = devices[loop]->channel;
    memcpy_s(msg.mac, WIFI_ANY_MAC_LEN, devices[loop]->bssid, WIFI_ANY_MAC_LEN);
    if (write_any_msg(&msg) != HI_ERR_SUCCESS) {
        printf("write scan result failed\r\n");
    }
    break;
}
printf("Total scanned ANY dev num: %d\r\n", num);
return;
}

int example_any_with_sta(void)
{
    hi_s32 ret;
    hi_u16 sleep_time_ms = 2000;
    hi_char ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};
    hi_s32 len = sizeof(ifname);
    hi_u8 sta_msg[] = "msg from sta";
    ret = hi_wifi_sta_start(ifname, &len);
    if (ret != HISI_OK) {
        return HISI_FAIL;
    }

    /* 初始化并注册发送完成和接收回调函数 */
    if (hi_wifi_any_init(ifname) == HISI_OK) {
        hi_wifi_any_set_callback(wifi_any_send_cb, wifi_any_recv_cb);
    }

    /* 创建任务用于处理ANY消息的收发 */
    if (any_start_callback_task() != HI_ERR_SUCCESS) {
        hi_wifi_any_set_callback(HI_NULL, HI_NULL);
        return hi_wifi_any_deinit();
    }
    strcpy((char *)g_any_msg, (char *)sta_msg);

    /* 扫描目标ANY设备 */
    while (!g_find_any_device) {
        printf("scanning any devices...\r\n");
        hi_wifi_any_discover_peer(wifi_any_scan_result_cb);
        hi_sleep(sleep_time_ms);
    }
    printf("STA and ANY start success!\r\n");
    return HISI_OK;
}

int example_any_with_ap(void)
{
    /* SoftAP接口的信息 */
    hi_wifi_softap_config hapd_conf = {
        "my_wifi", "", 1, 0, HI_WIFI_SECURITY_OPEN, HI_WIFI_PARIWISE_UNKNOWN};
    hi_char ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0}; /* 创建的SoftAP接口名 */
    hi_s32 len = sizeof(ifname); /* SoftAP接口名长度 */
    struct netif *netif_p = HI_NULL;
    ip4_addr_t st_gw;
    ip4_addr_t st_ipaddr;
```



```
ip4_addr_t st_netmask;

/* 这里用户配置自己的网关、IP、掩码 */
IP4_ADDR(&st_gw, 0, 0, 0, 0);
IP4_ADDR(&st_ipaddr, 0, 0, 0, 0);
IP4_ADDR(&st_netmask, 0, 0, 0, 0);
hi_u8 ap_msg[] = "msg from my_wifi";

/* 配置SoftAP网络参数， beacon周期修改为200ms */
if (hi_wifi_softap_set_beacon_period(200) != HISI_OK) {
    return HISI_FAIL;
}
/* 启动SoftAP接口 */
if (hi_wifi_softap_start(&hapd_conf, ifname, &len) != HISI_OK) {
    return HISI_FAIL;
}
/* 配置DHCP服务器 */
netif_p = netif_find(ifname);
if (netif_p == HI_NULL) {
    (hi_void)hi_wifi_softap_stop();
    return HISI_FAIL;
}

if (netifapi_netif_set_addr(netif_p, &st_ipaddr, &st_netmask, &st_gw) != HISI_OK) {
    (hi_void)hi_wifi_softap_stop();
    return HISI_FAIL;
}

if (netifapi_dhcps_start(netif_p, NULL, 0) != HISI_OK) {
    (hi_void)hi_wifi_softap_stop();
    return HISI_FAIL;
}

/* 初始化并注册发送完成和接收回调函数 */
if (hi_wifi_any_init(ifname) == HISI_OK) {
    hi_wifi_any_set_callback(wifi_any_send_cb, wifi_any_rcv_cb);
}
/* 创建任务用于处理ANY消息的收发 */
if (any_start_callback_task() != HI_ERR_SUCCESS) {
    hi_wifi_any_set_callback(HI_NULL, HI_NULL);
    return hi_wifi_any_deinit();
}
strcpy_s((char *)g_any_msg, ANY_MSG_MAX_SIZE, (char *)ap_msg);
printf("SOFTAP and ANY start success!\n");
return HISI_OK;
}

/* STA侧调用ANY Sample函数入口 */
hi_void app_main(hi_void)
{
    .....
    example_any_with_sta();
    .....
}

/* AP侧调用ANY Sample函数入口 */
hi_void app_main(hi_void)
{
    .....
    example_any_with_ap();
}
```



```
.....  
}
```

## 5.3 运行结果

ANY采用STA接口运行结果如下：

```
create any msg task success!  
scanning any devices...  
Total scanned ANY dev num: 0  
scanning any devices...  
Total scanned ANY dev num: 0  
scanning any devices...  
Total scanned ANY dev num: 0  
scanning any devices...  
Find ANY SSID: my_wifi, MAC: ac:11:31:a7:b4:4b Chan: 1 STA: N  
Total scanned ANY dev num: 1  
current channel 0 switch to channel 1  
{send to MAC:0x4b, status: 1 seq: 1}  
{rcv from MAC:0x4b, seq:2, len:16, data:msg from my_wifi}  
{send to MAC:0x4b, status: 1 seq: 3}  
{rcv from MAC:0x4b, seq:4, len:16, data:msg from my_wifi}  
{send to MAC:0x4b, status: 1 seq: 5}  
{rcv from MAC:0x4b, seq:6, len:16, data:msg from my_wifi}  
{send to MAC:0x4b, status: 1 seq: 7}  
{rcv from MAC:0x4b, seq:8, len:16, data:msg from my_wifi}  
{send to MAC:0x4b, status: 1 seq: 9}  
{rcv from MAC:0x4b, seq:10, len:16, data:msg from my_wifi}  
STA and ANY start success
```

对端ANY采用AP接口运行结果如下：

```
create any msg task success!  
SOFTAP and ANY start success!  
# {rcv from MAC:0x4a, seq:1, len:12, data:msg from sta}  
{send to MAC:0x4a, status: 1 seq: 2}  
{rcv from MAC:0x4a, seq:3, len:12, data:msg from sta}  
{send to MAC:0x4a, status: 1 seq: 4}  
{rcv from MAC:0x4a, seq:5, len:12, data:msg from sta}  
{send to MAC:0x4a, status: 1 seq: 6}  
{rcv from MAC:0x4a, seq:7, len:12, data:msg from sta}  
{send to MAC:0x4a, status: 1 seq: 8}  
{rcv from MAC:0x4a, seq:9, len:12, data:msg from sta}  
{send to MAC:0x4a, status: 1 seq: 10}
```



# 6 ANY 设备配对 Demo

## 6.1 概述

## 6.2 配对Demo实现方案

## 6.3 注意事项

## 6.1 概述

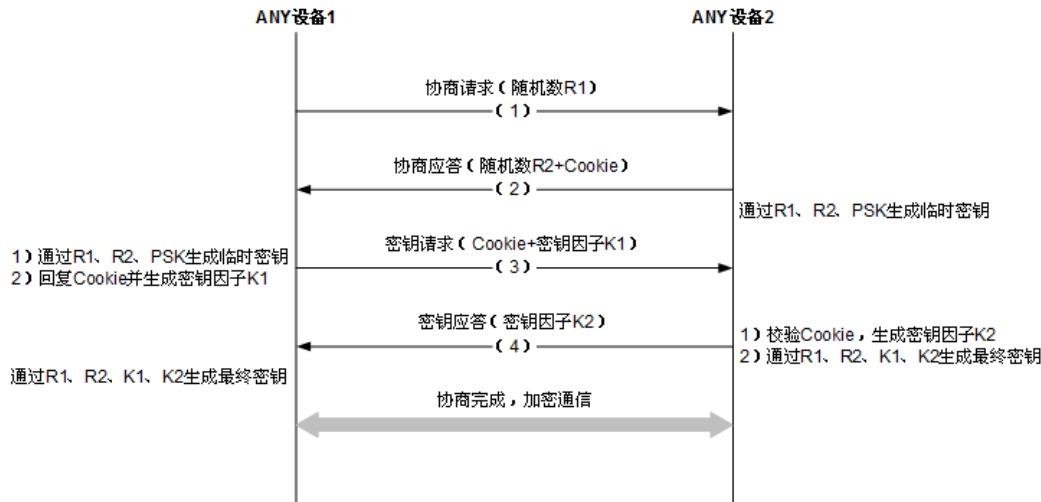
ANY设备支持加密通信，要求通信双方采用同样的16byte长度密钥进行加解密，一个密钥对应一个对端MAC地址，即采用该密钥加解密来自该MAC地址的ANY报文。用户应根据产品应用场景确定ANY设备之间如何产生和共享同样的密钥。SDK提供了ANY设备之间配对协商密钥的简单Demo示例（代码请参见demo/src/app\_any\_pair.c），供用户参考。该Demo实现了2个ANY设备采用ANY报文进行密钥协商，最终生成共同的密钥，用于后续的数据通信。

## 6.2 配对 Demo 实现方案

Demo采用图6-1实现方案，实现2个设备之间密钥协商，其中：设备1和设备2的配对协商消息均是采用ANY报文。协商请求和协商应答采用明文消息，协商请求消息中携带了随机数R1，协商应答消息携带了随机数R2和Cookie值。参与协商的2个设备需要采用同样的预共享密钥PSK，设备获取到随机数R1和R2之后结合PSK生成一个临时密钥用于加密密钥协商消息。设备1采用该临时密钥加密密钥请求消息，密钥请求消息中包括来自设备2的Cookie和随机生成的密钥因子K1。设备2采用临时密钥解密消息之后验证Cookie是否正确，验证通过则回复密钥应答消息。密钥应答消息中包含随机生成密钥因子K2，并采用临时密钥加密。双方完成上述交互之后，通过随机数R1和R2、密钥因子K1和K2采用同样的算法生成用于通信的最终密钥。



图 6-1 密钥协商流程图



### 6.3 注意事项

Demo仅供参考，只用于线下配对。当前IoT设备一般是通过路由器连接到云端，用户也可以结合网络应用场景设计实现其他安全的密钥生成和共享方案。