



Hi3861 V100 / Hi3861L V100 ANY Software

Development Guide

Issue	01
Date	2020-04-30

Copyright © HiSilicon (Shanghai) Technologies Co., Ltd. 2020. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon (Shanghai) Technologies Co., Ltd.

Trademarks and Permissions



HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided “AS IS” without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road,
Bantian, Longgang District,
Shenzhen 518129 P. R. China

Website: <https://www.hisilicon.com/en/>

Email: support@hisilicon.com



About This Document

Purpose

This document describes the application programming interfaces (APIs) and development processes of the ANY function of Hi3861 V100 and Hi3861L V100 Wi-Fi software.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3861	V100
Hi3861L	V100



Intended Audience

This document is intended for:




- Technical support engineers
- Software development engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
	Indicates a hazard with a medium level of risk which, if not avoided, could result in death or serious injury.



Symbol	Description
 CAUTION	Indicates a hazard with a low level of risk which, if not avoided, could result in minor or moderate injury.
 NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.
 NOTE	Supplements the important information in the main text. NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration.

Change History

Issue	Date	Change Description
01	2020-04-30	This issue is the first official release.
00B01	2020-04-03	This issue is the first draft release.



Contents

About This Document.....	i
1 Introduction.....	1
2 ANY Initialization and Deinitialization.....	2
2.1 Overview.....	2
2.2 Development Process.....	2
2.3 Precautions.....	3
2.4 Programming Example.....	4
3 ANY Device Scanning and Discovery.....	5
3.1 Overview.....	5
3.2 Development Process.....	5
3.3 Precautions.....	6
3.4 Programming Example.....	7
4 ANY Communication.....	8
4.1 Overview.....	8
4.2 Development Process.....	8
4.3 Precautions.....	10
4.4 Programming Example.....	10
5 ANY API Usage Example.....	11
5.1 Overview.....	11
5.2 Code Implementation.....	12
5.3 Running Result.....	18
6 ANY Device Pairing Demo.....	19
6.1 Overview.....	19
6.2 Pairing Demo.....	19
6.3 Precautions.....	20



1 Introduction

The ANY function is a Huawei proprietary short data communication function. It allows two Wi-Fi devices on the same channel to directly communicate with each other in point-to-point (P2P) connectionless mode. Hi3861 V100 and Hi3861L V100 provide ANY APIs for the app layer to implement the ANY application. The ANY APIs are used to initialize and deinitialize the ANY function, scan and discover ANY devices, and communicate with a peer ANY device. ANY can be used in wireless control scenarios such as smart bulb switch control, sensor data collection, and remote control for household appliances.

The ANY function has the following features:

- Each device can have a specific interface (for example, wlan0 or ap0) to send and receive ANY packets.
- ANY packets are sent and received through the channel where the interface is located. The interface and the peer device must be on the same channel.
- A single ANY packet supports a maximum of 250-byte data of the user layer.
- A single ANY device can communicate with a maximum of 16 ANY peer devices at the same time. Among them, a maximum of six ANY peer devices can be encrypted for communication.
- An ANY device can send and receive ANY unicast and broadcast packets instead of multicast packets.
- An ANY device can scan and discover other nearby ANY devices.

NOTE

For details about the APIs, see the *Hi3861 V100/Hi3861L V100 API Development Reference*.



2 ANY Initialization and Deinitialization

[2.1 Overview](#)

[2.2 Development Process](#)

[2.3 Precautions](#)

[2.4 Programming Example](#)

2.1 Overview

ANY initialization indicates that a created STA or SoftAP is selected as the interface for receiving and transmitting ANY packets on the local device, and the ANY function is enabled on the interface. After being initialized, ANY uses the channel of the selected interface to send and receive packets, and uses the MAC address of the interface as the source address and destination address.

- For a SoftAP, the channel is the one specified when the SoftAP is created.
- For an STA, if an STA is associated with a SoftAP, the channel of the STA is the one specified by the SoftAP. If the STA is not associated with any channel, the API function **hi_wifi_set_channel** may be used to specify a channel for the STA.

ANY deinitialization indicates that the ANY function is disabled. All ANY configurations, including the MAC address and communication key of the ANY peer device, are cleared after ANY deinitialization. If the ANY function has been initialized, you need to deinitialize it before reinitializing it.

2.2 Development Process

Application Scenario

To enable or disable the ANY function, the initialization and deinitialization APIs are needed.

Function

[Table 2-1](#) describes the ANY initialization and deinitialization APIs.



Table 2-1 Description of the ANY initialization and deinitialization APIs

API Name	Description
hi_wifi_any_init	Initializes the ANY function of an interface. The parameter is the interface name.
hi_wifi_any_deinit	Deinitializes the ANY function.

Development Process

The typical process of initializing and deinitializing the ANY application is as follows:

- Step 1** Start an STA by calling **hi_wifi_sta_start**. Or start a SoftAP by calling **hi_wifi_softap_start**.
- Step 2** Select an STA interface (wlan0) or an AP interface (ap0) as the ANY communication interface by calling **hi_wifi_any_init**.
- Step 3** Perform ANY scanning or ANY communication based on service requirements.
- Step 4** Deinitialize the ANY function by calling **hi_wifi_any_deinit**.

----End

Returns

[Table 2-2](#) describes the return values of the ANY initialization and deinitialization APIs.

Table 2-2 Return values of ANY initialization and deinitialization

No.	Definition	Actual Value	Description
1	HISI_OK	0	Operation succeeded.
2	HI_FAIL	1	Operation failed.

2.3 Precautions

- Only one interface can be selected as the ANY TX/RX interface for a device (that is, initialization is performed once). Before reinitialization, call the ANY deinitialization API.
- After the ANY is deinitialized, all ANY configurations are cleared.
- After being initialized on the STA or SoftAP interface, the ANY is automatically deinitialized when the STA or SoftAP is disabled.



2.4 Programming Example

For details about the example of ANY initialization on the STA interface, see the implementation of the **example_any_with_sta** function in [5 ANY API Usage Example](#).

For details about the example of ANY initialization on the SoftAP interface, see the implementation of the **example_any_with_ap** function in [5 ANY API Usage Example](#).



3 ANY Device Scanning and Discovery

[3.1 Overview](#)

[3.2 Development Process](#)

[3.3 Precautions](#)

[3.4 Programming Example](#)

3.1 Overview

After the ANY is initialized, you can use the scanning and discovery API to discover nearby devices that support the ANY and obtain information such as the device MAC address, channel, and RSSI strength.

3.2 Development Process

Application Scenario

After the ANY is initialized, the surrounding ANY devices are scanned.

Function

[Table 3-1](#) describes the ANY device scanning and discovery API.



Table 3-1 Description of the ANY device scanning and discovery API

API Name	Description
hi_wifi_any_discover_peer	Registers the callback function for scanning completion and performs an ANY scan. Before this function is called, you need to implement the scanning completion callback function of the hi_wifi_any_scan_result_cb type. For this callback function, the input arguments passed by the driver are the pointer array and the number of array elements of the discovered ANY device. Each element points to the information about a discovered ANY device.

Development Process

Devices can be discovered only after the ANY function is initialized. The development process is as follows:

- Step 1** Implement the **hi_wifi_any_scan_result_cb** callback function to process the result reported after the ANY scanning is complete.
- Step 2** Call **hi_wifi_any_discover_peer** to register the preceding callback function with the driver and start an ANY scan.

----End

Returns

[Table 3-2](#) describes the return values of the ANY device scanning and discovery API.

Table 3-2 Return values of the ANY device scanning and discovery API

No.	Definition	Actual Value	Description
1	HISI_OK	0	Operation succeeded.
2	HI_FAIL	1	Operation failed.

3.3 Precautions

- After the ANY is initialized, an STA or SoftAP can perform ANY scanning and discover other surrounding ANY devices. In addition, the STA and SoftAP can be scanned by other ANY devices.
- In the scanning result, if the discovered ANY device is a SoftAP, the device information contains the SSID information. If the discovered ANY device is an STA, the SSID is not contained (null string).



- The callback function runs on a driver thread and cannot be blocked or wait for a long time. You are advised to create a task to process the scanning result. After the callback function copies the result to the task, the callback function exits.
- Information about a maximum of 32 peer devices can be returned in a single scan by using the callback function. The array memory passed by the callback function is managed by the driver and should not be released in the callback function.

3.4 Programming Example

For details about the example of initiating an ANY scan, see the implementation of **example_any_with_sta** in [5 ANY API Usage Example](#). If the target device is not found in the example, the scan continues.

For details about how to compile the callback function after scanning is complete, see the implementation of **wifi_any_scan_result_cb** in [5 ANY API Usage Example](#). In the example, if the target device is matched, the message processing task is notified to initiate ANY communication to the device. You can also allocate memory in the callback function for scanning completion, copy all scanning results, send them to the message processing task for processing, and then exit the task quickly.



4 ANY Communication

[4.1 Overview](#)

[4.2 Development Process](#)

[4.3 Precautions](#)

[4.4 Programming Example](#)

4.1 Overview

An ANY device can communicate with other ANY devices on the same channel in encryption or non-encryption mode. A single ANY packet supports a maximum of 250 bytes of user data. If encrypted communication is required, both communication parties must be configured with the same 16-byte key in advance. You need to implement the receive callback function to process the received message from the peer end and implement the transmit completion callback function. After sending an ANY message to the peer end, the driver calls this callback function to check whether the message is successfully sent (whether the ACK packet from the peer end is received).

4.2 Development Process

Application Scenario

After the ANY is initialized, it communicates with the peer ANY device on the same channel.

Function

[Table 4-1](#) describes the ANY communication APIs.



Table 4-1 Description of the ANY communication APIs

API Name	Description
hi_wifi_any_set_callback	Registers the callback functions for receiving and sending ANY packets before communication. You need to implement the callback function for receiving hi_wifi_any_rcv_cb packets. After this callback function is registered, the driver calls this function to transmit the received any packets to the app layer. You can implement the callback function of the hi_wifi_any_send_complete_cb type. Each time the transmission is complete, the driver calls this function to feed back the transmission result (whether the transmission is successful and the ACK message is received) to the app layer. If the transmission is successful, the value of status is 1 . If this function is not required, the value is registered as NULL .
hi_wifi_any_add_peer	Adds information about the ANY peer device, including the MAC address and key used for encrypted communication with the peer device. The same key must be configured for the peer device to parse encrypted data.
hi_wifi_any_del_peer	Deletes information about the peer device with a specified MAC address.
hi_wifi_any_fetch_peer	Queries information about the peer device with a specified index. Generally, the information about all configured peer devices can be queried.
hi_wifi_any_send	Sends ANY data to the peer device with a specified MAC address. The data size cannot exceed 250 bytes each time.

Development Process

The ANY communication can be performed only after the ANY function is initialized. Before the communication, ensure that the local end and the peer end are in the same channel. The development process is as follows:

- Step 1** Implement the **hi_wifi_any_rcv_cb** and **hi_wifi_any_send_complete_cb** callback functions.
- Step 2** Call **hi_wifi_any_set_callback** to register the preceding callback functions with the driver.
- Step 3** Call **hi_wifi_any_add_peer** to configure the key and bind it to the MAC address of the peer device. If the key is not configured, the communication is in plaintext.
- Step 4** Call **hi_wifi_any_send** to send data to the peer device. If the MAC address of the peer device is bound to a key, the key is used to encrypt the communication data.

----End



Returns

[Table 4-2](#) describes the return values of the ANY communication APIs.

Table 4-2 Return values of the ANY communication APIs

No.	Definition	Actual Value	Description
1	HISI_OK	0	Operation succeeded.
2	HI_FAIL	1	Operation failed.

4.3 Precautions

- The callback function for receiving and transmission completion runs on a driver thread and cannot be blocked or wait for a long time. You are advised to create a task to process received and transmitted packets, copy data to the new task in the callback function, and exit the task.
- An ANY device can communicate with a maximum of 16 peers and can communicate with a maximum of six peers in encrypted mode. The same key must be configured for the two communication ends.
- The data memory passed by the driver to the callback receiving function is managed by the driver. The memory should not be freed in the callback receiving function.
- The ANY transmits and receives data only on the current channel where the interface is located. If the peer end is on another channel, the channel switching API needs to be called to switch the channel before communication.

4.4 Programming Example

For the example of the communication between two ANY devices, see [5 ANY API Usage Example](#).



5 ANY API Usage Example

5.1 Overview

5.2 Code Implementation

5.3 Running Result

5.1 Overview

This example implements P2P communication between two ANY devices (ANY STA and ANY SoftAP).

- An ANY STA is created in the entry function **example_any_with_sta** and the ANY function is initialized on the STA interface. In addition, a message processing task is created to send and receive ANY messages and process scanning results. Then ANY scanning is performed to search for the target ANY device.
- An ANY SoftAP is created in the entry function **example_any_with_ap** and the DHCP server starts. Then, the ANY function is initialized on the SoftAP interface and a message processing task is created to receive and send ANY messages.

NOTICE

In the implementation of **wifi_any_rcv_cb** for receiving messages and **wifi_any_send_cb** for completing message sending, to prevent the driver thread from being blocked for a long time, the system exits after sending messages to the message processing task through simple copying. During compilation and running, **example_any_with_sta** can be mounted to **app_main** of the STA device, and **example_any_with_ap** can be mounted to **app_main** of the SoftAP device.

NOTE

In this example, the communication is in plaintext. The same 16-byte communication key can be configured for the communication ends through **hi_wifi_any_add_peer** before the communication. In this way, the communication can be encrypted.

5.2 Code Implementation

NOTE

In this example, ANY STA and ANY SoftAP can share all code. For ANY STA, call the **example_any_with_sta** entry function in **app_main**. For ANY SoftAP, call the **example_any_with_ap** entry function.

The following is a code sample for ANY initialization, scanning, and communication:

```
#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;
    /* Receive frames with the same sequence number. By default, they are retransmission frames
at the app layer and directly return. The external caller frees the memory. */
    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: format;
5: index*/
    for (unsigned short loop = 0; loop < len; loop++) {
        printf("%c", data[loop]);
    }
    printf(" }\r\n");

    /* Reply a message to the peer end as a sample to reply a limited number of messages. */
    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)
{
    /* Obtain the channel of the current device. For example, the interface name is wlan0 when
the ANY scanning is initiated by an STA and ap0 when the ANY scanning is initiated by an AP. */
    hi_u8 uc_len = strlen("wlan0") + 1;
    hi_s32 current_channel = hi_wifi_get_channel("wlan0", uc_len);
    /*Switch to another channel to send ANY data.*/
    /*Note: If services are running on the current channel (for example, the current channel is
associated with a router), switching to another channel will interrupt the services.*/
    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);
    }

    /* Send data to the scanned target ANY device. The TX sequence number of the frame is set
to 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_rcv_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;
    }

    /* Initialize and register the callback functions for sending and receiving data. */
    if (hi_wifi_any_init(ifname) == HISI_OK) {
        hi_wifi_any_set_callback(wifi_any_send_cb, wifi_any_rcv_cb);
    }

    /*Create a task for receiving and sending ANY messages.*/
    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);

    /*Scan the target ANY device.*/
    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 interface information*/
    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}; /* Name of the SoftAP interface*/
hi_s32 len = sizeof(ifname); /* Length of the SoftAP interface name*/
struct netif *netif_p = HI_NULL;
ip4_addr_t st_gw;
ip4_addr_t st_ipaddr;
ip4_addr_t st_netmask;

/*Configure the user's gateway, IP address, and subnet mask.*/
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";

/*Configure the SoftAP network parameters and set the beacon interval to 200 ms.*/
if (hi_wifi_softap_set_beacon_period(200) != HISI_OK) {
    return HISI_FAIL;
}
/*Start the SoftAP*/
if (hi_wifi_softap_start(&hapd_conf, ifname, &len) != HISI_OK) {
    return HISI_FAIL;
}
/*Configure the DHCP server.*/
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;
}

/* Initialize and register the callback functions for sending and receiving data. */
if (hi_wifi_any_init(ifname) == HISI_OK) {
    hi_wifi_any_set_callback(wifi_any_send_cb, wifi_any_rcv_cb);
}
/*Create a task for receiving and sending ANY messages.*/
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;
}

/* Entry for the STA to call the ANY sample function*/
hi_void app_main(hi_void)
{
    ...
    example_any_with_sta();
    ...
}
```



```
/* Function entry for the AP to call the ANY sample*/  
hi_void app_main(hi_void)  
{  
    ...  
    example_any_with_ap();  
    ...  
}
```

5.3 Running Result

The following code shows the running result when the ANY uses the STA interface:

```
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}  
{recv from MAC:0x4b, seq:2, len:16, data:msg from my_wifi }  
{send to MAC:0x4b, status: 1 seq: 3}  
{recv from MAC:0x4b, seq:4, len:16, data:msg from my_wifi }  
{send to MAC:0x4b, status: 1 seq: 5}  
{recv from MAC:0x4b, seq:6, len:16, data:msg from my_wifi }  
{send to MAC:0x4b, status: 1 seq: 7}  
{recv from MAC:0x4b, seq:8, len:16, data:msg from my_wifi }  
{send to MAC:0x4b, status: 1 seq: 9}  
{recv from MAC:0x4b, seq:10, len:16, data:msg from my_wifi }  
STA and ANY start success
```

The following code shows the running result when the peer ANY uses the AP interface:

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



6 ANY Device Pairing Demo

[6.1 Overview](#)

[6.2 Pairing Demo](#)

[6.3 Precautions](#)

6.1 Overview

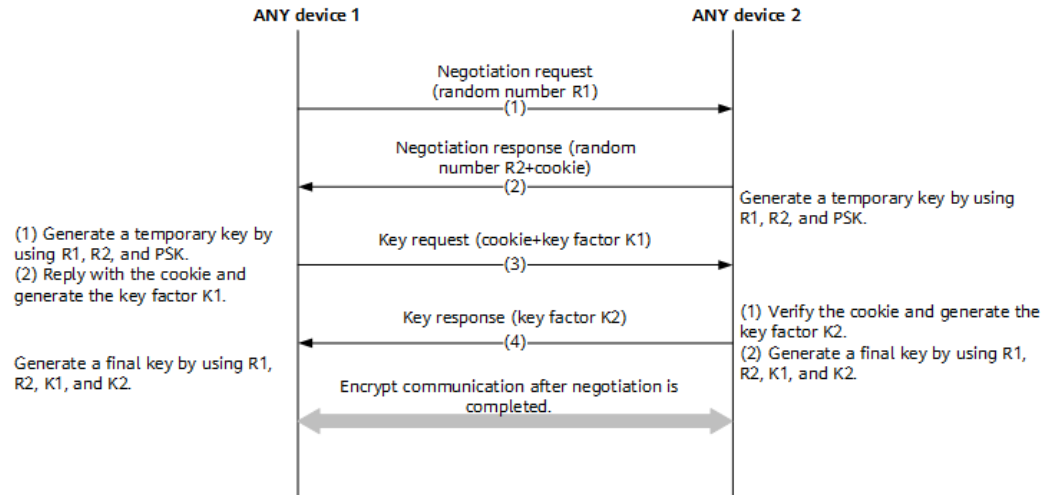
An ANY device supports encrypted communication. The two communication ends must use the same 16-byte key for encryption and decryption. One key corresponds to one peer MAC address. That is, the key is used to encrypt and decrypt ANY packets from the MAC address. You need to determine how to generate and share the same key between any devices based on the application scenario. The SDK provides a simple demo sample for negotiating keys between ANY devices. For details about the code, see **demo/src/app_any_pair.c**. In this demo, two ANY devices use ANY packets to negotiate the key and generate the same key for subsequent data communication.

6.2 Pairing Demo

The demo uses the implementation scheme in [Figure 6-1](#) to implement key negotiation between two devices. The pairing negotiation messages of device 1 and device 2 are ANY packets. The negotiation request and the negotiation response are plaintext messages. The negotiation request carries a random number R1, and the negotiation response carries a random number R2 and a cookie value. The two devices participating in the negotiation need to use the same pre-shared key PSK. After obtaining the random numbers R1 and R2, the device generates a temporary key based on the PSK to encrypt the key negotiation message. Device 1 encrypts the key request message by using the temporary key, where the key request message includes a cookie from device 2 and a randomly generated key factor K1. After decrypting the message by using the temporary key, device 2 verifies whether the cookie is correct. If the verification succeeds, a key response message is returned. The key response message contains a randomly generated key factor K2 and is encrypted by using a temporary key. After completing the interactions, the two ends generate a final key used for

communication by using the random numbers R1 and R2 and the key factors K1 and K2 based on the same algorithm.

Figure 6-1 Key negotiation process



6.3 Precautions

The demo is for reference only and is used only for offline pairing. Currently, IoT devices are connected to the cloud through routers. You can design other secure key generation and sharing solutions based on network application scenarios.