ESP-NOW

Table des matières

Overview

ESP-NOW is a kind of connectionless Wi-Fi communication protocol that is defined by Espressif. In ESP-NOW, application data is encapsulated in a vendor-specific action frame and then transmitted from one Wi-Fi device to another without connection.

CTR with CBC-MAC Protocol (CCMP) is used to protect the action frame for security. ESP-NOW is widely used in smart light, remote controlling, sensor, etc.

Frame Format

ESP-NOW uses a vendor-specific action frame to transmit ESP-NOW data. The default ESP-NOW bit rate is 1 Mbps. The format of the vendor-specific action frame is as follows:

MAC Header	Category Code	Organization Identifier	Random Values	Vendor Specific Content
24 bytes	1 byte	3 bytes	4 bytes	7-257 bytes

- Category Code: The Category Code field is set to the value (127) indicating the vendor-specific category.
- Organization Identifier: The Organization Identifier contains a unique identifier (0x18fe34), which is the first three bytes of MAC address applied by Espressif.
- Random Value: The Random Value filed is used to prevents relay attacks.
- Vendor Specific Content: The Vendor Specific Content contains vendor-specific fields as follows:

Element ID	Length	Organization Identifier	Type	Version	Body
1 byte	1 byte	3 bytes	1 byte	1 byte	0-250 bytes

- Element ID: The Element ID field is set to the value (221), indicating the vendor-specific element.
- Length: The length is the total length of Organization Identifier, Type, Version and Body.
- Organization Identifier: The Organization Identifier contains a unique identifier (0x18fe34), which is the first three bytes of MAC address applied by Espressif.
- Type: The Type field is set to the value (4) indicating ESP-NOW.
- Version: The Version field is set to the version of ESP-NOW.
- Body: The Body contains the ESP-NOW data.

As ESP-NOW is connectionless, the MAC header is a little different from that of standard frames. The FromDS and ToDS bits of FrameControl field are both 0. The first address field is set to the destination address. The second address field is set to the source address. The third address field is set to broadcast address (0xff:0xff:0xff:0xff:0xff:0xff).

Security

ESP-NOW uses the CCMP method, which is described in IEEE Std. 802.11-2012, to protect the vendor-specific action frame. The Wi-Fi device maintains a Primary Master Key (PMK) and several Local Master Keys (LMK). The lengths of both PMK and LMk are 16 bytes.

- PMK is used to encrypt LMK with the AES-128 algorithm. Call esp_now_set_pmk() to set PMK. If PMK is not set, a default PMK will be used.

- LMK of the paired device is used to encrypt the vendor-specific action frame with the CCMP method. The maximum number of different LMKs is six. If the LMK of the paired device is not set, the vendor-specific action frame will not be encrypted.

Encrypting multicast vendor-specific action frame is not supported.

Initialization and Deinitialization

Call `esp_now_init()` to initialize ESP-NOW and `esp_now_deinit()` to de-initialize ESP-NOW. ESP-NOW data must be transmitted after Wi-Fi is started, so it is recommended to start Wi-Fi before initializing ESP-NOW and stop Wi-Fi after de-initializing ESP-NOW.

When `esp_now_deinit()` is called, all of the information of paired devices are deleted.

Add Paired Device

Call `esp_now_add_peer()` to add the device to the paired device list before you send data to this device. If security is enabled, the LMK must be set. You can send ESP-NOW data via both the Station and the SoftAP interface. Make sure that the interface is enabled before sending ESP-NOW data.

The maximum number of paired devices is 20, and the paired encryption devices are no more than 17, the default is 7. If you want to change the number of paired encryption devices, set `CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM` in the Wi-Fi component configuration menu.

A device with a broadcast MAC address must be added before sending broadcast data. The range of the channel of paired devices is from 0 to 14. If the channel is set to 0, data will be sent on the current channel. Otherwise, the channel must be set as the channel that the local device is on.

Send ESP-NOW Data

Call `esp_now_send()` to send ESP-NOW data and `esp_now_register_send_cb()` to register sending callback function. It will return `ESP_NOW_SEND_SUCCESS` in sending callback function if the data is received successfully on the MAC layer. Otherwise, it will return `ESP_NOW_SEND_FAIL`. Several reasons can lead to ESP-NOW fails to send data. For example, the destination device does not exist; the channels of the devices are not the same; the action frame is lost when transmitting on the air, etc. It is not guaranteed that application layer can receive the data. If necessary, send back ack data when receiving ESP-NOW data. If receiving ack data timeouts, retransmit the ESP-NOW data. A sequence number can also be assigned to ESP-NOW data to drop the duplicate data.

If there is a lot of ESP-NOW data to send, call esp_now_send() to send less than or equal to 250 bytes of data once a time. Note that too short interval between sending two ESP-NOW data may lead to disorder of sending callback function. So, it is recommended that sending the next ESP-NOW data after the sending callback function of the previous sending has returned. The sending callback function runs from a high-priority Wi-Fi task. So, do not do lengthy operations in the callback function. Instead, post the necessary data to a queue and handle it from a lower priority task.

Receiving ESP-NOW Data

Call `esp_now_register_recv_cb()` to register receiving callback function. Call the receiving callback function when receiving ESP-NOW. The receiving callback function also runs from the Wi-Fi task. So, do not do lengthy operations in the callback function. Instead, post the necessary data to a queue and handle it from a lower priority task.

Config ESP-NOW Rate

Call `esp_wifi_config_espnow_rate()` to config ESP-NOW rate of specified interface. Make sure that the interface is enabled before config rate. This API should be called after `esp_wifi_start()`.

Config ESP-NOW Power-saving Parameter

Sleep is supported only when ESP32 is configured as station.

Call `esp_now_set_wake_window()` to configure Window for ESP-NOW RX at sleep. The default value is the maximum, which allowing RX all the time.

If Power-saving is needed for ESP-NOW, call `esp_wifi_connectionless_module_set_wake_interval()` to configure Interval as well.

ESP-NOW One-Way Communication

For example, in one-way communication, you can have scenarios like this:

One ESP32 board sending data to another ESP32 board

This configuration is very easy to implement and it is great to send data from one board to the other like sensor readings or ON and OFF commands to control GPIOs.

A “master” ESP32 sending data to multiple ESP32 “slaves”

One ESP32 board sending the same or different commands to different ESP32 boards. This configuration is ideal to build something like a remote control. You can have several ESP32 boards around the house that are controlled by one main ESP32 board.

One ESP32 “slave” receiving data from multiple “masters”

This configuration is ideal if you want to collect data from several sensors nodes into one ESP32 board. This can be configured as a web server to display data from all the other boards, for example.

ESP-NOW Two-Way Communication

With ESP-NOW, each board can be a sender and a receiver at the same time. So, you can establish two-way communication between boards.

For example, you can have two boards communicating with each other.

Source : https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/

ESP32: Getting Board MAC Address

To communicate via ESP-NOW, you need to know the MAC Address of the ESP32 receiver. That's how you know to which device you'll send the data to.

Each ESP32 has a unique MAC Address and that's how we identify each board to send data to it using ESP-NOW (learn how to Get and Change the ESP32 MAC Address).

```
\ change this parameters with your own wifi params
z" Mariloo"                       constant mySSID
z" 1925144D91DE5373C3C2D7959F"  constant myPASSWORD

wifi

\ Initialize WiFi
WIFI_MODE_STA Wifi.mode
mySSID myPASSWORD WiFi.begin

create mac 6 allot          \ store local MAC address
mac WiFi WiFi.macAddress

\ display MAC address in hex format
: .mac ( -- addr len )
    base @ hex
    6 0 do
        mac i + c@ <# # # #> type
        i 5 < if
            [char] : emit
        then
    loop
    base !
  ;
.mac
```

Functions list

esp_now_init (--)

esp_err_t esp_now_init(void);

Initialize ESPNOW function

return :

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_INTERNAL : Internal error

esp_now_deinit (--)

esp_err_t esp_now_deinit(void);

De-initialize ESPNOW function

return :

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_INTERNAL : Internal error

esp_now_get_version (addr -- fl)

esp_err_t esp_now_get_version(uint32_t *version);

Get the version of ESPNOW

@param

- version ESPNOW version

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_ARG : invalid argument

esp_now_register_recv_cb (xt -- fl)

esp_err_t esp_now_register_recv_cb(esp_now_recv_cb_t cb);

Register callback function of receiving ESPNOW data.

This function is part of the ESP-NOW library and is crucial for setting up a callback mechanism to handle incoming data from paired ESP devices.

A callback function is a custom function you define that gets executed whenever new data arrives through ESP-NOW.

Define the Callback Function

This function will be called whenever data is received via ESP-NOW. It typically takes three arguments:

- uint8_t *aa (sender's MAC address): This points to an array containing the MAC address of the device that sent the data.
- uint8_t *data (received data): This points to the actual data buffer that holds the received information.
- int len (data length): This indicates the size (in bytes) of the received data.

```
void onDataRecv(const uint8_t *aa, const uint8_t *data, int len) {
    // Your code to process the received data (e.g., print, store, etc.)
    Serial.print("Received data from: ");
    Serial.println(aa, HEX);
    Serial.print("Data: ");
    for (int i = 0; i < len; i++) {
        Serial.print(data[i], HEX);
        Serial.print(" ");
    }
    Serial.println();
}
```

Use esp_now_register_recv_cb to associate your custom callback function with ESP-NOW for receiving data:

```
esp_now_register_recv_cb(onDataRecv);
```

Call this line within your code's `setup()` function to register the callback before starting ESP-NOW communication.

Complete example

```
#include <esp_now.h>

// Replace with your sender's MAC address
uint8_t peer_addr[ESP_NOW_PEER_ADDR_LEN] = {0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX};

void onDataRecv(const uint8_t *aa, const uint8_t *data, int len) {
    Serial.print("Received data from: ");
    Serial.println(aa, HEX);
    Serial.print("Data: ");
    for (int i = 0; i < len; i++) {
        Serial.print(data[i], HEX);
        Serial.print(" ");
    }
    Serial.println();
}

void setup() {
    Serial.begin(115200);

    // Initialize ESP-NOW
    esp_now_init();

    // Add peer device (replace with actual MAC address)
    esp_now_add_peer(peer_addr);

    // Register callback for receiving data
    esp_now_register_recv_cb(onDataRecv);
}

void loop() {
```

```
    // Add your main program loop here (optional)
}
```

Explanation

- Replace 0xXX in peer_addr with the actual MAC address of the ESP device you want to receive data from.
- The onDataRecv function demonstrates processing the received data by printing it to the serial monitor. You can adapt this to your specific application logic.
- Call esp_now_init() and esp_now_add_peer() to initialize ESP-NOW and add the peer device before registering the callback.

@param

- cb callback function of receiving ESPNOW data

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_INTERNAL : internal error

esp_now_unregister_recv_cb (--)

esp_err_t esp_now_unregister_recv_cb(void);

Unregister callback function of receiving ESPNOW data,

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

esp_now_register_send_cb (xt -- fl)

esp_err_t esp_now_register_send_cb(esp_now_send_cb_t cb);

Register callback function of sending ESPNOW data

@param

- cb callback function of sending ESPNOW data

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_INTERNAL : internal error

esp_now_unregister_send_cb

esp_err_t esp_now_unregister_send_cb(void);

Unregister callback function of sending ESPNOW data.

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

esp_err_t esp_now_send (peer_addr data len --)

esp_err_t esp_now_send(const uint8_t *peer_addr, const uint8_t *data, size_t len);

Send ESPNOW data

If peer_addr is not NULL, send data to the peer whose MAC address matches peer_addr

If peer_addr is NULL, send data to all of the peers that are added to the peer list

The maximum length of data must be less than ESP_NOW_MAX_DATA_LEN

The buffer pointed to by data argument does not need to be valid after esp_now_send returns

@param

- peer_addr peer MAC address
- data data to send
- len length of data

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument
  - ESP_ERR_ESPNOW_INTERNAL : internal error
  - ESP_ERR_ESPNOW_NO_MEM : out of memory, when this happens, you can delay a while before sending the next data
  - ESP_ERR_ESPNOW_NOT_FOUND : peer is not found
  - ESP_ERR_ESPNOW_IF : current Wi-Fi interface doesn't match that of peer
    ESP_ERR_ESPNOW_CHAN: current Wi-Fi channel doesn't match that of peer

esp_now_add_peer(peer – fl)

esp_err_t esp_now_add_peer(const esp_now_peer_info_t *peer);

Add a peer to peer list

@param

- peer peer information

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument
  - ESP_ERR_ESPNOW_FULL : peer list is full
  - ESP_ERR_ESPNOW_NO_MEM : out of memory
  - ESP_ERR_ESPNOW_EXIST : peer has existed

esp_now_del_peer(peer_addr – fl)

esp_err_t esp_now_del_peer(const uint8_t *peer_addr);

Delete a peer from peer list

@param

- peer_addr peer MAC address

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument
  - ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

esp_err_t esp_now_mod_peer(peer – fl)

esp_err_t esp_now_mod_peer(const esp_now_peer_info_t *peer);

Modify a peer

@param

- peer peer information

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument
  - ESP_ERR_ESPNOW_FULL : peer list is full

esp_now_set_peer_rate_config (peer_addr config)

esp_err_t esp_now_set_peer_rate_config(const uint8_t *peer_addr, esp_now_rate_config_t *config);

Set ESPNOW rate config for each peer

This API should be called after esp_wifi_start() and esp_now_init().

@param

- peer_addr peer MAC address
- config rate config to be configured.

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument
  - ESP_ERR_ESPNOW_INTERNAL : internal error

esp_now_get_peer (addr peer – fl)

esp_err_t esp_now_get_peer(const uint8_t *peer_addr, esp_now_peer_info_t *peer);

Get a peer whose MAC address matches peer_addr from peer list.

@param

- peer_addr peer MAC address
- peer peer information

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument
  - ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

esp_now_fetch_peer (from_head peer – fl)

esp_err_t esp_now_fetch_peer(bool from_head, esp_now_peer_info_t *peer);

Fetch a peer from peer list. Only return the peer which address is unicast, for the multicast/broadcast address, the function will ignore and try to find the next in the peer list.

@param

- from_head fetch from head of list or not
- peer peer information

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument
  - ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

esp_now_is_peer_exist (peer_addr – fl)

bool esp_now_is_peer_exist(const uint8_t *peer_addr);

Peer exists or not

@param

- peer_addr peer MAC address

@return

- - true : peer exists
  - false : peer not exists

esp_now_get_peer_num (num – fl)

esp_err_t esp_now_get_peer_num(esp_now_peer_num_t *num);

Get the number of peers

@param

- num number of peers

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument

esp_now_set_pmk (pmk – fl)

esp_err_t esp_now_set_pmk(const uint8_t *pmk);

Set the primary master key

@param

- pmk primary master key

primary master key is used to encrypt local master key

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG : invalid argument

esp_now_set_wake_window (window – fl)

esp_err_t esp_now_set_wake_window(uint16_t window);

Set wake window for esp_now to wake up in interval unit

@param

- window Milliseconds would the chip keep waked each interval, from 0 to 65535.

This configuration could work at connected status.

When ESP_WIFI_STA_DISCONNECTED_PM_ENABLE is enabled, this configuration could work at disconnected status.

Default value is the maximum.

@return

- - ESP_OK : succeed
  - ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized