# 智能配置通信协议

从这几天的调试来看，cc3200不支持混杂模式，而无线网络的数据包iP层全部是加密的（除非没密码），可见的802.11的MAC头，结合组播的特点，可以将数据加载到IP地址上，即以组播地址为载体，通过改变组播地址将SSID与密钥数据发送出去。为此设计如下协议：

1. **报文格式**

一个组播包一次只能发送23位的数据，为此将包做如下划分：

1. **DEVID包[I]：**

特点低16位装载DEVID（DEVID为二进制格式即16个字节，而非字符串格式即32个字节），高7位用于序号标记，值有127到121用标记设备ID的高14字节。此类包在数据长度包的前面发送。（明文）

Destination MAC:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 0x01 | 0x00 | 0x5E | 0x7f-0x79 | DEVID[x] | DEVID[x] |

1. **信标包[B]：**

特点组播的IP固定，低16位为设备ID的低26位，高7位固定为120，并且每隔三个数据包就插发一个信标包。作用用于设备的信道锁定，并初步判断app是否在绑定它自己。（明文）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 0x01 | 0x00 | 0x5E | 0x78 | DEVID[1] | DEVID[0] |

1. **数据长度包[L]：**

特点用来指数据包的个数，高7位，固定为119，中间8位固定为255，低8位即数据包长度值的两倍。值的范围0～118，即数据包的最大长度为238。在数据包的前面和尾部发送。(明文)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 0x01 | 0x00 | 0x5E | 0x77 | 0xff | Data\_len |

1. **数据包[D]：**

特点高7位为数据报的序号（0～118），低16位为数据。（密文\明文）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 0x01 | 0x00 | 0x5E | 0x00-0x76 | Data1 | Data2 |

1. **序列规律**

将要发送的数据装载在组播包的ip地址中，并以如下序列发送：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **B** | **I[1]** | **I[2]** | **I[3]** | **B** | **I[4]** | **I[5]** | **I[6]** | **B** | **I[7]** | **L** | **D[1]** | **B** | **D[2]** | **D[3]** | **D[4]** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **B** | **D[5]** | **D[6]** | **D[7]** | **B** | **…** | **D[N]** | **L** | **B** | **I[1]** | **I[2]** | **I[3]** | **…** |

1. **数据报文的格式（明文）：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | Key\_type | 1 | Key\_len | 1 | ssid\_len |
| 0xfa | 0(none)1(wep)2(wpa) | 0xfb | Key | 0xfc | SSID |

对整个数据报文进行des加密。

加解密的函数如下：

说明：密钥采用devid字符串形式以4的倍数为下标的新字符串。

例如：01234567890123456789012345678901

0 4 8 2 6 0 4 8

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//! brief DES encrypt method:ecb

//! param[in] plaintext

//! param[in] len(value-result):input plaintext-len,output ciphertext-len

//! param[in] devid(len=32): original key

//! return ciphertext-call malloc,so users need free

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint8\_t \***DES\_encrypt**(**const** uint8\_t \*plaintext,**int** \*len,**const** uint8\_t \*devid)

{

DES\_key\_schedule ks;

uint8\_t devid\_hex[16];

uint8\_t okey[8];

uint8\_t tmp[8];

uint8\_t \*ciphertext;

**int** size = \*len;

**int** i = 0;

**int** count = 0;

**if**(devid == NULL || (**strlen**(devid) !=32)) **return** NULL;

strtohex(devid\_hex,devid);

**memcpy**(okey, &devid\_hex[6], 8); // devid\_hex(6~13)be used as original key

DES\_set\_key\_unchecked((const\_DES\_cblock\*)okey, &ks);

size = (size / 8 + (size % 8 ? 1: 0)) \* 8;

ciphertext = **malloc**(size);

**if**(ciphertext == NULL) **return** NULL;

**memset**(ciphertext,0,size);

**memcpy**(ciphertext, plaintext,\*len);

/\* 分组加密，每8字节一组 \*/

count = size / 8;

**for** (i = 0; i < count; i++)

{

**memcpy**(tmp, ciphertext + 8 \* i, 8);

DES\_ecb\_encrypt((const\_DES\_cblock\*)tmp, (DES\_cblock\*)(ciphertext + 8 \* i), &ks, DES\_ENCRYPT);

}

\*len = size;

**return** ciphertext;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//! brief DES decrypt method:ecb

//! param[in] ciphertext

//! param[in] len(value-result):input ciphertext-len,output plaintext-len

//! param[in] devid(len=32): original key

//! return plaintext-call malloc,so users need free

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint8\_t \***DES\_decrypt**(**const** uint8\_t \*ciphertext,**int** \*len,**const** uint8\_t \*devid)

{

DES\_key\_schedule ks;

uint8\_t devid\_hex[16];

uint8\_t okey[8];

uint8\_t tmp[8];

uint8\_t \*plaintext;

**int** size = \*len;

**int** i = 0;

**int** count = 0;

strtohex(devid\_hex,devid);

**memcpy**(okey, &devid\_hex[6], 8); // devid\_hex(6~13)be used as original key

DES\_set\_key\_unchecked((const\_DES\_cblock\*)okey, &ks);

**if**((size == 0)||(size % 8)) **return**;

plaintext = **malloc**(size);

**if**(plaintext == NULL) **return** NULL;

/\* 分组解密，每8字节一组 \*/

count = size / 8;

**for** (i = 0; i < count; i++) {

**memcpy**(tmp, ciphertext + 8 \* i, 8);

DES\_ecb\_encrypt((const\_DES\_cblock\*)tmp, (DES\_cblock\*)(plaintext + 8 \* i), &ks, DES\_DECRYPT);

}

**return** plaintext;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//! strtohex

//! \brief Convert str to hex

//! \param[in] hex,Str,Strlen

//! \return dest.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**static** **void** \***strtohex**(**void** \*hex, **const** **void** \*Str)

{

uint32\_t charCnt;

uint32\_t flag = 0;

**if**(hex == NULL || Str == NULL) **return** NULL;

**char** \*phex = hex;

**char** \*pStr = (**char** \*)Str;

uint32\_t Strlen = **strlen**(Str);

// Start from end

pStr += Strlen;

\*phex = 0x00;

**for** (charCnt = 0; charCnt < Strlen; charCnt++)

{

pStr--;

**switch**(\*pStr)

{

**case** '0':

\*phex |= 0x00;

**break**;

**case** '1':

\*phex |= 0x10;

**break**;

**case** '2':

\*phex |= 0x20;

**break**;

**case** '3':

\*phex |= 0x30;

**break**;

**case** '4':

\*phex |= 0x40;

**break**;

**case** '5':

\*phex |= 0x50;

**break**;

**case** '6':

\*phex |= 0x60;

**break**;

**case** '7':

\*phex |= 0x70;

**break**;

**case** '8':

\*phex |= 0x80;

**break**;

**case** '9':

\*phex |= 0x90;

**break**;

**case** 'A':

**case** 'a':

\*phex |= 0xa0;

**break**;

**case** 'B':

**case** 'b':

\*phex |= 0xb0;

**break**;

**case** 'C':

**case** 'c':

\*phex |= 0xc0;

**break**;

**case** 'D':

**case** 'd':

\*phex |= 0xd0;

**break**;

**case** 'E':

**case** 'e':

\*phex |= 0xe0;

**break**;

**case** 'F':

**case** 'f':

\*phex |= 0xf0;

**break**;

**default**:

**break**;

}

**if**(flag)

{

\*++phex = 0x00;

}

**else**

{

\*phex = (\*phex) >> 4;

\*phex &= 0x0000000f;

}

flag = 1 - flag;

}

**return** hex;

}

**测试代码：**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//! brief DES-ECB加密方式测试

//! param[in] argc argv

//! return 0 OK -1 err (gcc -o des\_ecb des\_ecb\_test.c -lcrypto)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int** **main**(**int** argc, uint8\_t \*argv[])

{

uint8\_t \*devid = "f50c93a980ad497999eb44124bfd1adf";

uint8\_t \*otext = "longwe1234";

uint8\_t \*ciphertext;

uint8\_t \*plaintext;

**int** len;

**int** i;

len = **strlen**(otext)+1;

ciphertext = DES\_encrypt(otext,&len,devid);

**if**(ciphertext == NULL)

{

**printf**("encrypt err!\n");

**return** -1;

}

**printf**("len= %d\n",len);

**printf**("encrypt message:\n");

**for** (i = 0; i < len; i++)

{

**printf**("%02x", \*(ciphertext + i));

}

**printf**("\n");

plaintext = DES\_decrypt(ciphertext,&len,devid);

**if**(plaintext == NULL)

{

**printf**("decrypt err!\n");

**return** -1;

}

**printf**("decrypt message:\n");

**for** (i = 0; i < len; i++)

{

**printf**("%02x", \*(plaintext + i));

}

**printf**("\n");

**printf**("plaintext = %s\n",plaintext);

**free**(ciphertext);

**free**(plaintext);

**return** 0;

}

# 绑定流程

用户扫码

设备启动

生成组播编码序列

(携带devid、ssid、pwd、pwd\_t)

802.11信道监听

信道锁定

广播config

间隔1ms依次发送组播序列

收到config应答？

NO

YES

组播解码

连接ap、获取IP

局域网TCP/UDP，连云TCP启动

解码OK？

NO

YES

收到confing立即应答

发送bind

获取token

等待、超时重复

收到bind

发送devowner

响应bind

收到bind应答，立即发送ACK

Bind成功

Bind成功

Conifg：携带服务器地址、app版本与时区。