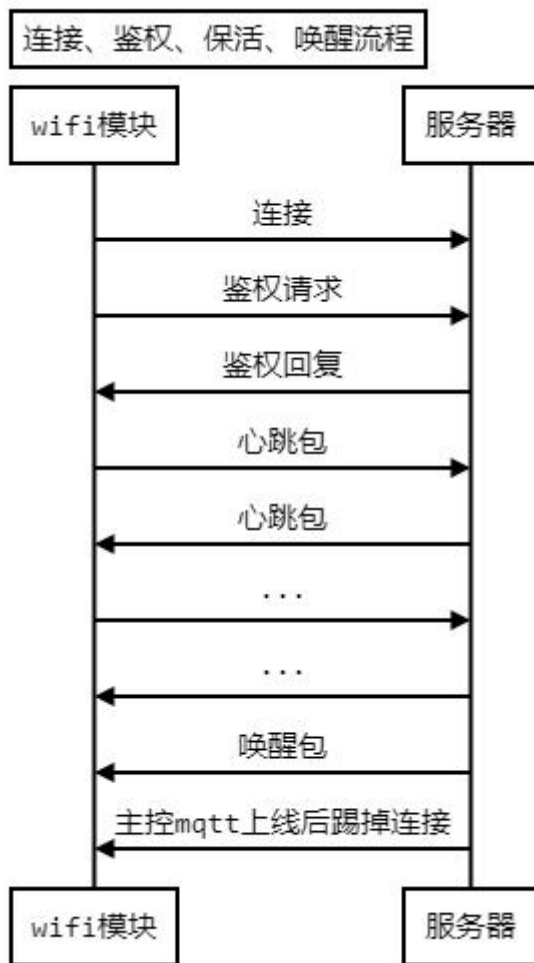


低功耗唤醒协议开发文档

涂鸦 IPC 为客户提供两种形式的低功耗保活方法。一种用户参考低功耗协议与云端对接。一种是使用涂鸦 IPC 封装好的低功耗协议接口与云端对接。本文针对第一种方式使用进行说明。

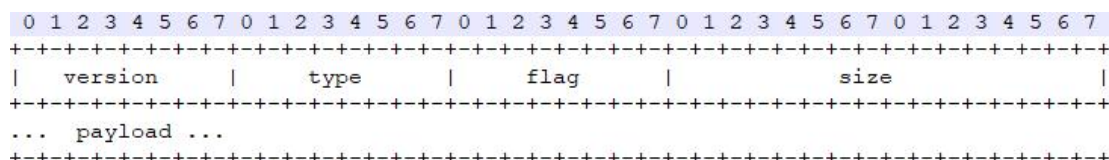
低功耗唤醒服务功能，在通信方面，主要包含三个方面的数据通信（默认大端传输）：

- 1、认证，构建起数据通道（通过 `tuya_ipc_low_power_server_get` 获得服务器 ip 和 443 端口，`tuya_ipc_device_id_get` 获取 devId 、`tuya_ipc_local_key_get` 获取加密 key）
- 2、心跳数据发送；
- 3、唤醒数据包。



一：通用数据格式：数据头+payload

version	type	flag	size	payload
---------	------	------	------	---------



Wifi 模组与服务器之间的数据通信统一采用上图的格式，由 header 和 payload 组成，其中 header 包括：

Version: 1 字节，协议版本号，第一版写 1

Type: 1 字节，命令号，0：鉴权请求，1：鉴权回复，2：心跳包，3：唤醒包

名称	数值	意义
LP_TYPE_AUTH_REQUEST	0	ipc向服务器发起认证请求
LP_TYPE_AUTH_RESPONSE	1	服务器回复ipc认证请求
LP_TYPE_HEARTBEAT	2	心跳类型
LP_TYPE_WAKEUP	3	唤醒类型

Flag: 1 字节，标识符，高 4 位保留（用于以后的扩展），低 4 位用于表征数据段加密类型，初始定义认证交互的`payload`字段采用`aes cbc 128`加密方式，采用`PKCS7Padding`方式填充，`flag`为`0x1`，低 4 位若是为`0`则表示不加密

Size: 2 字节，`payload`字段总的长度，不包含`version`、`type`、`flag`三个字段的长度，该字段是否需要加密根据`flag`类别定

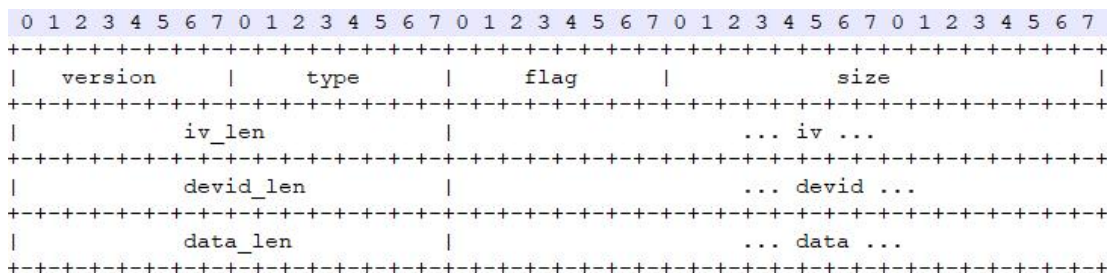
payload: 加载的具体数据，详情见后面解析

二：鉴权过程

Wifi 模组和服务器之间建立 tcp 连接后，鉴权过程由 wifi 模组先发起请求，服务器对请求数据进行校验，通过后回复一个报文，wifi 模组再进行签名校验，整个过程完成后即建立了一个数据通道，可以进行后续心跳包的发送。

鉴权的数据交互采用前述的通用数据格式，payload 部分包括 iv, devid, data 三部分，以上三个字段采用 LV 形式组织，即长度+实际有效数据。

ivlen	iv	devIdlen	devId	dataLen	data
-------	----	----------	-------	---------	------



iv: 用于解密使用，用户自定义随机生成一个 16 字节数据；

devid: 加密后的设备标识，服务器用来查询解密需要使用的 key；原始 devid 使用 SDK 接口 `tuya_ipc_device_id_get()` 获取（**devidlen 不能为 0 长度不小于 26**），鉴于安全考虑，原始 devid 需要使用 aes cbc 加密（**aes_cbc 需要做 pkcs7 的 padding**）再进行 base64 加密，加密使用的 key 和 iv 固定如下：

```
key[16] = {0x23, 0xac, 0x7b, 0x15, 0x0d, 0x89, 0x34,
            0x92, 0xf1, 0x19, 0x33, 0xde, 0xc8, 0x6a,
            0x10, 0x55};
```

```
iv[16] = {0x1e, 0x25, 0x77, 0xb8, 0x66, 0xc1, 0x10,
           0x33,
           0x93, 0x69, 0xcb, 0xa8, 0x2c, 0x54, 0xe5, 0xab};
```

data: 交互数据，需要采用 aes cbc 128 加密；**data 的加密使用 localkey 和 payload 里的 iv**

Wifi 模组向服务器发起握手请求，data 参数格式如下：

```
{
  "type": 1,
  "method": 1,
  "authorization": "time=138993455,random=sdfsijjwejwemkejitejwitetopwejriew",
  "signature": "i2je8fjsesfjeijfiejwifheuhfuwefjweif",
}
```

type: 表示发起请求的是wifi 模组 0，还是tuya_SDK 1；固定为1

method: 签名算法类型，默认为 1；

authorization: “time: 设备的 UTC 时间，服务器不做校验，Random: 随机的 32 字节 “

signature: 按照 devid:time:random 格式拼接，对拼接后的数据先进行 hmac sha256 加密，然后进行 base 64 加密，计算得出。（`tuya_ipc_local_key_get` 接口获取加密 key）

注意：这里 devid 也是需要 encoded devid,参考：

```
cnXNEYU1PDSVTurAXMzK7m6u+1QQsHu9rwQZUSutSbU=:1618796830
;ke_fZa[UOVPJDLE?GA;4<6071uovpj
```

服务器消息返回：服务器按照签名算法计算出 signature，同时生成相应的参数返回给设备，服务器返回的格式如下：

```
{
  "err": 0,
  "interval": 60,
  "random": "sdfsiiweiwemkeiitejwitetopweiriew",
  "authorization": "time=138993455,random=quertyuikfhkof18458yeiurur",
  "signature": "i2je8fjsesfjeijfiejwifheuhfuwefjweif",
}
```

err: 错误码， 0 表示成功，其它表示失败；
interval: 心跳间隔时间；固定的，改不了
random: IPC 带给服务器的随即数，服务器原样返回；
authorization: 认证参数，包含时间和随机数；
signature: 签名数据；

IPC 对接受到的数据进行校验认证：

1. 使用云端返回 payload 中的 iv 和 localkey 对 data 进行 aes decode 解密
2. 将 random 和请求中的 random 进行比较，若不一致则校验失败
3. 将 authrization 中的 time 和 random 使用与请求中 devid:time:random 相同的方式计算签名
4. 将上一步计算出的签名与报文中的 signature 进行对比，若不一致则校验失败
5. 注意逗号不需要参与计算，不要用字符串解析 json，注意字段反序列化实现。

三：心跳数据

心跳数据内容固定，没有 payload，第一版 version 为 1， type 为 2， flag 为 0， size 字段为 0，最后组装的数据如下：

0x1, 0x2, 0x0, 0x0, 0x0

四：唤醒数据

唤醒数据包，第一版 version 为 1， type 为 3， flag 为 0， size 字段为 4， payload 字段由 sdk 提供 `tuya_ipc_local_key_get()` 获取 local key，通过计算 local key 的 hash crc32 值而来。

0x1, 0x3, 0x0, 0x0, 0x4, 0x11, 0x23, 0xab, 0xbf