蓝牙门锁小程序 SDK

智能硬件解决方案 > 门锁 > RN 面板和小程序开发

文档版本: 20201226

查看在线版本

目录

本文介绍了微信小程序和涂鸦蓝牙门锁的通信过程，以及给出相关的示例代码供开发者参考使用。本文相关代码仅适用于 3.* 及以上版本的使用涂鸦 BLE 通信协议进行通信的设备。通信数据单位为字节（Byte）。

1 蓝牙 SDK

涂鸦提供蓝牙 SDK，根据涂鸦蓝牙协议提供处理数据的方法和配置。

1.1 安装 SDK

需要在 `.npmrc` 文件中配置对应的 registry：

```
@tuya-wx:registry=https://registry-out-npm.tuya-inc.top/
```

使用 NPM 包管理器下载 SDK。

```
npm install @tuya-wx/ble-sdk
```

1.2 发送或接收数据

小程序与蓝牙门锁的通信建立在数据接收、发送的基础上，开发者需要自行封装接收、发送函数。本文中提供数据发送方法`sendBle`以及数据接收的示例代码以供参考。

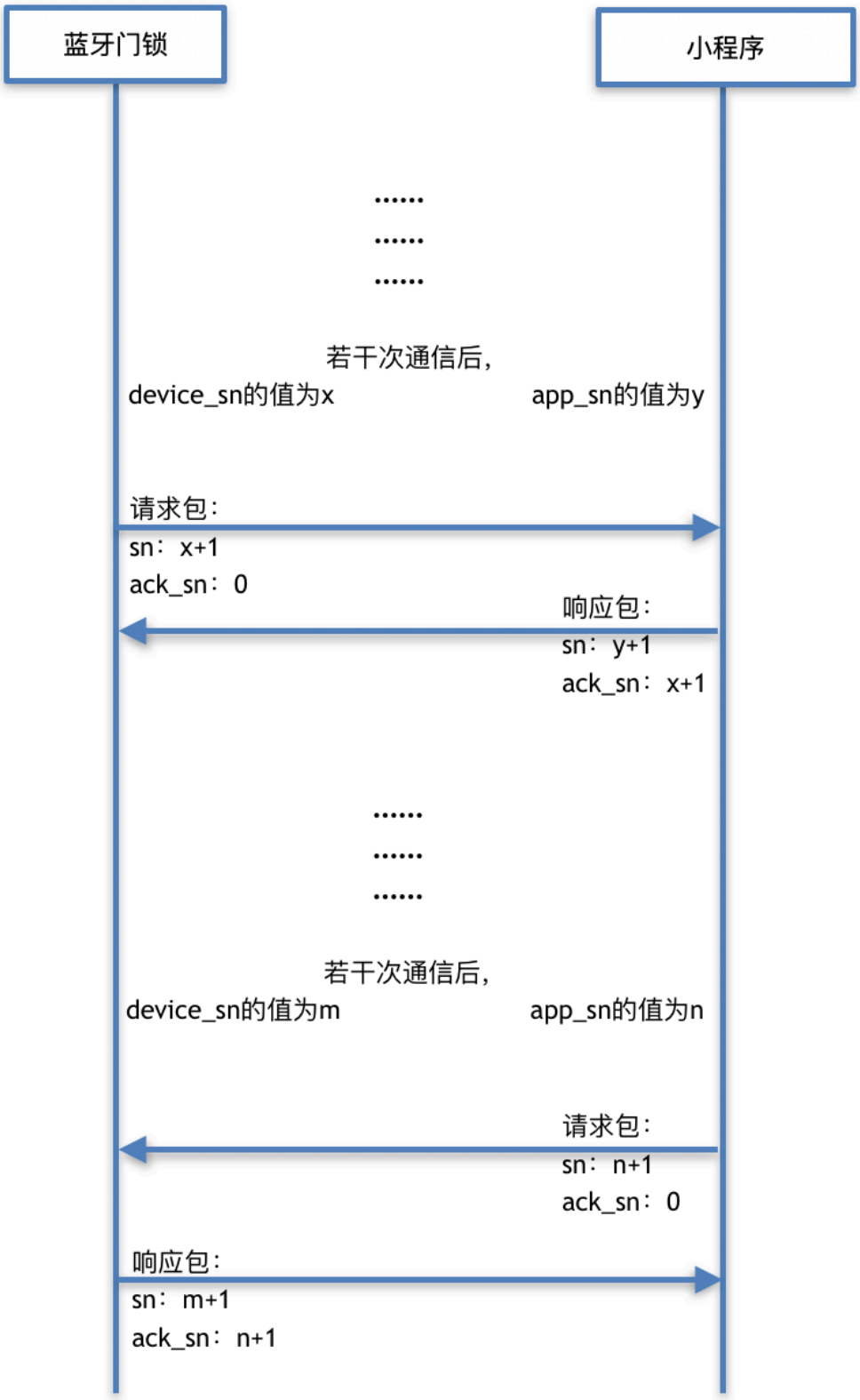
1.3 数据结构

蓝牙门锁通信时，使用涂鸦 BLE 通信协议 3.* 版本，数据结构如下：

字段	长度	说明
sn	4	请求包序列号
ack_sn	4	响应包序列号
code	2	功能码
data_len	2	数据长度 len
data	len	数据

- sn：小程序和设备端各维护自己的 sn，每次新的连接初始值为 1，每发送一包指令依次滚动加 1，溢出复位重新赋值 1。
- ack_sn：只有响应包的 ACK_SN 才有意义，代表是响应的哪条指令，不是响应包默认为 0。

  请求包序列号 sn、响应包序列号 ack_sn 变化见下图。


蓝牙门锁
小程序
.....
.....
.....
若干次通信后，
device_sn的值为x
app_sn的值为y
请求包：
sn：x+1
ack_sn：0
响应包：
sn：y+1
ack_sn：x+1
.....
.....
.....
若干次通信后，
device_sn的值为m
app_sn的值为n
请求包：
sn：n+1
ack_sn：0
响应包：
sn：m+1
ack_sn：n+1

{width=400px}

- data：传输数据，具体见下文功能码描述。

2 发送数据

小程序发送数据，需要封装发送 API，API 内将较长的数据分片发送。

发送 API 可以参考 `sendBle`，使用示例及源码如下。

2.1 `sendBle` 源码

```
import { parseData, config, util } from '@tuya-wx/ble-sdk';
export const sendBle = (sn, ackSn, code, data, datalength, flagKey, key, random) => {
  console.log('【发送】' + code.toString(16) + explainCode(code) + ' sn:' + sn + ' ackSn:' + ackSn);
  // 封装命令
  const raw = parseData.commandGenerate(sn, ackSn, code, data, datalength);
  // 对数据进行打包和加密
  const command = parseData.packetData(key, flagKey, random, raw);
  return sendCommandBle(command);
};
export const sendCommandBle = async command => {
  // 对数据传输进行分包
  const packetToData = parseData.packet(config.Coder.VERSION2, command, command.length);
  // deviceId、serviceUuid、characteristicIdUuid通信建立时会获取，具体参考下文“和蓝牙门锁建立连接”中的“建立通信连接”
  const serviceUuid = Taro.getStorageSync('serviceUuid');
  const deviceId = Taro.getStorageSync('deviceId');
  const characteristicIdUuid = Taro.getStorageSync('characteristicIdUuid');
  // 写入数据
  const canWriteFlag =
    serviceUuid && characteristicIdUuid && Array.isArray(packetToData) && packetToData.length > 0;
  if (!canWriteFlag) return { success: false, msg: '数据错误，无法发送' };
  let errFlag;
  await packetToData.forEach(async data => {
    try {
      await Taro.writeBLECharacteristicValue({
        deviceId: deviceId,
        serviceId: serviceUuid,
        characteristicId: characteristicIdUuid,
        value: new Uint8Array(data).buffer,
      });
    } catch (err) {
      errFlag = err;
      console.log('【发送错误】', err);
    }
  });
  if (errFlag) return { success: false, msg: errFlag.errMsg };
  console.log('【发送】end');
  return { success: true };
};
```

2.2 参数说明

字段	数据类型	说明
sn	number	请求包序列号
ackSn	number	响应包序列号
code	number	功能码
data	array[number]	数据
datalength	number	数据长度 data.length
flagKey	number	数据加密方式
key	string	加密密钥
random	string	登入密钥

> **说明**：发送数据需要使用加密密钥 `key` 和登入密钥 `random` 进行加密后进行传输，接收的数据同样需要加密密钥和登入密钥进行解密。

密钥 key 名称	密钥 flag (flagKey)	生成方式
key1	FLAG_1_KEY1	从云端获取，参考下文配对密钥
key4	FLAG_4_KEY4	MD5(login_key)
key5（即 sessionKey）	FLAG_5_KEY5	MD5(login_key,dev_rand)

具体的密钥获取步骤，请参考下文示例代码。

2.3 使用示例

```
sendBle(
  4, // sn
  2, // ack_sn
  0x0002, // code
  [26, 1, 1, 0], // data
  4, // data_len
  config.SecurityFlag.FLAG_5_KEY5, // 加密flag
  sessionKey, // 加密密钥
  random, // 登入密钥
);
```

3 接收数据

蓝牙门锁分片发送数据，小程序调用 onBLECharacteristicValueChange 监听蓝牙特征值改变，接收并进行消息组装。

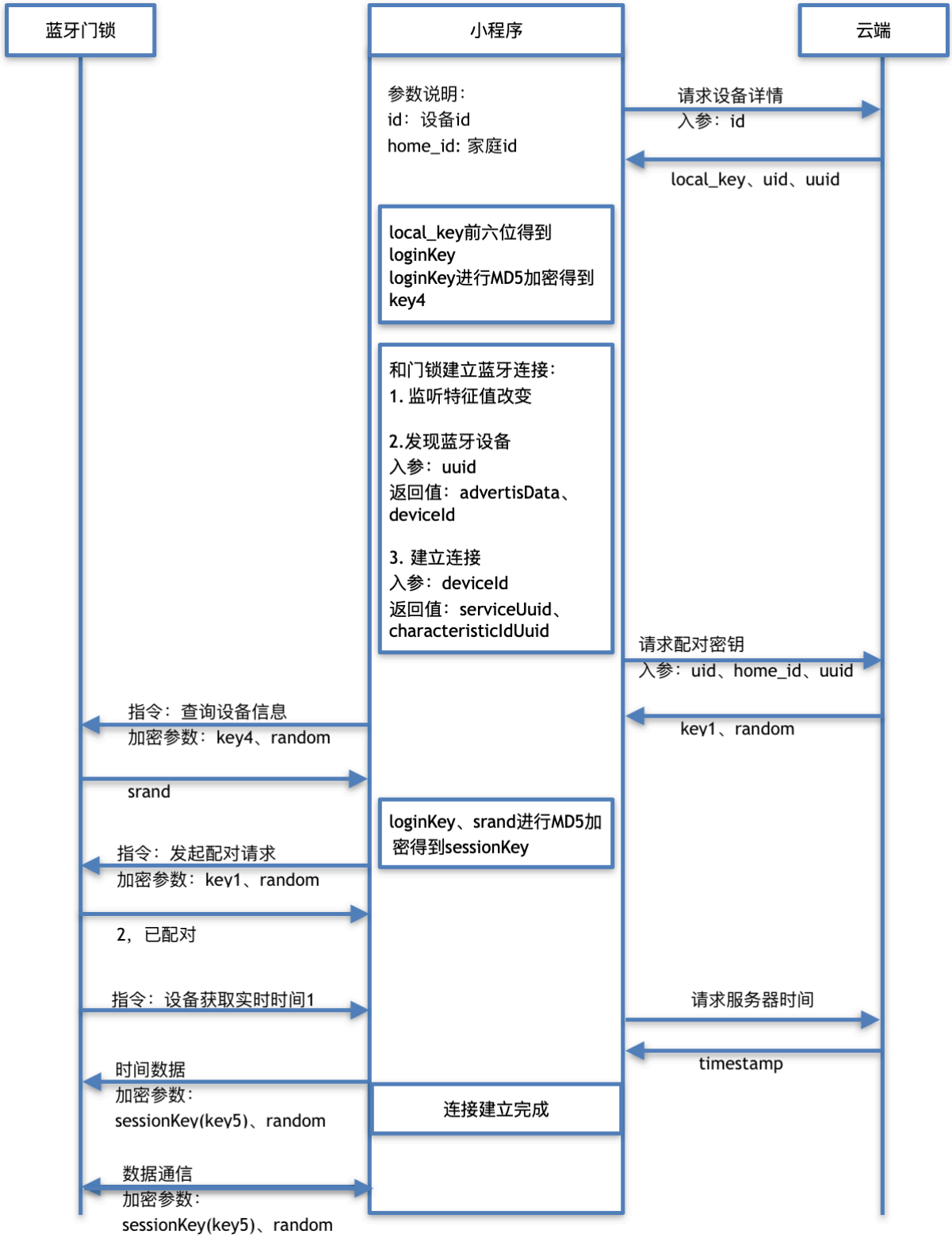
3.1 接收代码示例

```
import { parseData, config, util } from '@tuya-wx/ble-sdk';
const { MTP_CONTINUE, MTP_OK, MTP_FAIL, MTP_LENGTH_FAIL } = util.MIT
_STATUS;
const { SecurityFlag } = config;
// 监听特征值改变
wx.onBLECharacteristicValueChange(res => {
  const { value } = res;
  const str = util.ab2hex(value).join('');
  const key = util.HexString2Bytes(str);
  parseDataReceived(key, dataReducer);
});
// 对蓝牙特征值比特数据进行处理
const parseDataReceived = (() => {
  let receiveData = [];
  let receiveObjData = {};
  return (receive, reducer) => {
    // 解析每一帧数据 每一帧 20 byte
    const { status, data } = parseData.unpack(receive);
    // 存储帧数据
    if (status === MTP_CONTINUE || status === MTP_OK || status === M
TP_LENGTH_FAIL)
      receiveData.push(...data);
    if (status !== MTP_OK) return null;
    // 帧数据接收完毕
    receiveObjData = { version: 0, raw: receiveData };
    // 数据重置
    receiveData = [];
    if (receiveObjData.raw == null || receiveObjData.raw.length == 0
) return receiveObjData;
    const flag = parseData.receiveFlag(receiveObjData.raw); // 加密fla
g
    const key1 = Taro.getStorageSync('encryptedAuthKey'); // secret_
key_1
    const loginKey = Taro.getStorageSync('loginKey'); // login_key
    const srand = Taro.getStorageSync('srand'); // dev_rand
    let key = [];
    switch (flag) {
      case SecurityFlag.FLAG_1_KEY1:
        key = util.HexString2Bytes(key1);
        break;
      case SecurityFlag.FLAG_4_KEY4:
        key = parseData.getSecretKey4(loginKey);
        break;
      case SecurityFlag.FLAG_5_KEY5:
        key = parseData.getSessionKey(loginKey, srand);
        break;
    }
    const { sn, sn_ack, code, b_data } = parseData.parseReceiveData(
key, receiveObjData.raw, flag);
    Taro.setStorageSync('sn_dev', sn); // 存储设备端sn
    Taro.setStorageSync('sn_ack_dev', sn_ack); // 存储设备端ack_sn
    reducer(sn, sn_ack, code, b_data);
    return receiveObjData;
  };
})();
// 数据处理函数，针对收到的不同code和 data作出处理
const dataReducer = (sn, sn_ack, code, data) => {
  switch (code) {
    case config.FUN_SENDER.FUN_SENDER_DEVICE_INFO:
      // ...
```

4 建立通信

当小程序使用涂鸦配网插件进行蓝牙门锁配网之后，从设备列表页进入蓝牙门锁面板，小程序需要重新与门锁进行蓝牙配对，建立通信。

4.1 通信时序图


蓝牙门锁
小程序
云端
参数说明：
id：设备id
home_id: 家庭id
请求设备详情
入参：id
local_key、uid、uuid
local_key前六位得到loginKey
loginKey进行MD5加密得到key4
和门锁建立蓝牙连接：
1. 监听特征值改变
2.发现蓝牙设备
入参：uuid
返回值：advertisData、deviceId
3. 建立连接
入参：deviceId
返回值：serviceUuid、characteristicIdUuid
请求配对密钥
入参：uid、home_id、uuid
key1、random
指令：查询设备信息
加密参数：key4、random
srand
loginKey、srand进行MD5加密得到sessionKey
指令：发起配对请求
加密参数：key1、random
2，已配对
指令：设备获取实时时间1
请求服务器时间
timestamp
时间数据
加密参数：
sessionKey(key5)、random
连接建立完成
数据通信
加密参数：
sessionKey(key5)、random

4.2 示例代码

```
import { util, parseData, config } from '@tuya-wx/ble-sdk';
```

设备详情接口获取参数

用设备 id 从设备详情接口获取参数 local_key, uid, uuid，从而计算得出 loginKey、key4。

```
const res = await getDeviceDetails(id);
const { local_key, uid, uuid } = res;
const loginKey = local_key.substr(0, 6);
const key4 = parseData.getSecretKey4(loginKey);
wx.setStorageSync('loginKey', loginKey);
wx.setStorageSync('secretKey4', key4);
```

和蓝牙门锁建立连接

从广播设备中找到目标蓝牙门锁，获取广播数据 advertisData。

```
await wx.openBluetoothAdapter();
await wx.startBluetoothDevicesDiscovery();
const { devices } = await wx.getBluetoothDevices(); // 可使用该API轮询
    搜索结果
wx.stopBluetoothDevicesDiscovery();
let devData;
devices.forEach(item => {
  let { advertisServiceUUIDs, serviceData, advertisData } = item;
  serviceData = util.ab2hex(serviceData[advertisServiceUUIDs[0]]);
  advertisData = util.ab2hex(advertisData);
  const pData = parseData.getUuid(advertisData, serviceData);
  if (uuid === pData.uuid)
    devData = { ...item, serviceData, advertisData, uuid, productKey
: pData.productKey };
  // item 中包含 deviceId
});
return devData;
```

建立通信连接

```
Taro.onBLECharacteristicValueChange(res => {
  const { value } = res;
  const str = util.ab2hex(value).join('');
  const key = util.HexString2Bytes(str);
  parseDataReceived(key, dataReducer);
});
wx.createBLEConnection({ deviceId });
const { services } = await wx.getBLEDeviceServices({ deviceId });
const service = services[0];
const { characteristics } = await wx.getBLEDeviceCharacteristics({
  deviceId,
  serviceId: service.uuid,
});
const connectData = {};
characteristics.forEach(item => {
  if (item.properties.notify) {
    connectData['Notify'] = { characteristicIdUuid: item.uuid, servi
ceUuid: service.uuid };
    Taro.notifyBLECharacteristicValueChange({
      deviceId,
      serviceId: service.uuid,
      characteristicId: item.uuid,
      state: true,
    });
  } else if (item.properties.write) {
    connectData['Write'] = { characteristicIdUuid: item.uuid, servic
eUuid: service.uuid };
  }
});
const { characteristicIdUuid, serviceUuid } = connectData['Write'];
Taro.setStorageSync('deviceId', deviceId);
Taro.setStorageSync('serviceUuid', serviceUuid);
Taro.setStorageSync('characteristicIdUuid', characteristicIdUuid);
```

发起配对请求

获取 key1 和 random。

```
const TYPE_FLAG = parseData.getCompId(advertisData);
if (TYPE_FLAG !== 4) return; //不支持的compId
const adData = parseData.parseRadioData(advertisData);
if (adData.protocolVersion !== 3) return; // 非 协议版本3,不支持
if (adData.encryptFlag !== 0 && adData.encryptFlag !== 3) return; //
 不支持的编码方式
if (!adData.registerStyle) return; //非 通过ble注册,不支持
// 向云端请求配对密钥
const res = await pairingAuthKey({ uid, home_id, uuid });
Taro.setStorageSync('encryptedAuthKey', res.encrypted_auth_key); //
key1
Taro.setStorageSync('random', res.random); // random
```

向设备发送信息：查询设备信息。

```
const random = Taro.getStorageSync('random');
const sn = Taro.getStorageSync('sn_app');
sendBle(
  +sn + 1,
  0,
  config.FUN_SENDER.FUN_SENDER_DEVICE_INFO,
  null,
  0,
  config.SecurityFlag.FLAG_4_KEY4,
  Taro.getStorageSync('secretKey4'),
  util.HexString2Bytes(random),
);
Taro.setStorageSync('sn_app', +sn + 1);
```

收到查询设备信息的响应后：

1. 得到 srand 和 sessionKey
2. 发起配对请求

```
const dataReducer = async (sn, sn_ack, code, data) => {
  console.log(`【接收】${code.toString(16)}:${explainCode(code)} sn:${sn
} sn_ack:${sn_ack}`);
  console.log('【接收】数据: ' + data.toString());
  switch (code) {
    case config.FUN_SENDER.FUN_SENDER_DEVICE_INFO:
      // 得到 srand 和 sessionKey
      this.resetSrandAndSessionKey(data);
      // 发起配对请求
      this.send2pair();
      break;
    // ...
    // ...
    // ...
    default:
      console.log('暂不支持的指令', code);
      break;
  }
};
const resetSrandAndSessionKey = data => {
  const { srand } = parseData.parseSearchDeviceBData(data);
  Taro.setStorageSync('srand', srand);
  const loginKey = Taro.getStorageSync('loginKey');
  const session_key = parseData.getSessionKey(loginKey, srand);
  Taro.setStorageSync('sessionKey', session_key);
};
const send2pair = async () => {
  let input = util.stringToByte(uuid);
  const loginKey = Taro.getStorageSync('loginKey');
  input = [...input, ...util.stringToByte(loginKey), ...util.stringT
oByte(id)];
  const len = 44 - input.length;
  for (let i = 0; i < len; i++) input.push(0x00);
  let key1 = Taro.getStorageSync('encryptedAuthKey');
  const random = Taro.getStorageSync('random');
  const sn = Taro.getStorageSync('sn_app');
  sendBle(
    +sn + 1,
    0,
    config.FUN_SENDER.FUN_SENDER_PAIR,
    input,
    input.length,
    config.SecurityFlag.FLAG_1_KEY1,
    util.HexString2Bytes(key1),
    util.HexString2Bytes(random),
  );
  Taro.setStorageSync('sn_app', +sn + 1);
};
```

收到“配对请求”的响应后，小程序会再次收到“设备获取实时时间 1”的请求，并作出响应。

```
const dataReducer = async (sn, sn_ack, code, data) => {
  console.log(`【接收】${code.toString(16)}:${explainCode(code)} sn:${sn
    sn
} sn_ack:${sn_ack}`);
  console.log('【接收】数据：' + data.toString());
  switch (code) {
    case config.FUN_SENDER.FUN_SENDER_PAIR:
      // 配对请求的响应：2，已配对
      console.log('配对结果', code);
      break;
    case config.FUN_SENDER.FUN_RECEIVE_TIME1_REQ:
      // 响应“设备获取实时时间1”
      responseTime();
      break;
    // ...
    // ...
    // ...
    default:
      console.log('暂不支持的指令', code);
      break;
  }
};
const responseTime = async () => {
  const time = await getCurrentTime(); // 接口获取云端时间
  const newTime = time
    .toString()
    .split('')
    .map(num => num.charCodeAt());
  newTime.push(3, 32);
  const snApp = Taro.getStorageSync('sn_app');
  const snDev = Taro.getStorageSync('sn_dev');
  const random = util.HexString2Bytes(Taro.getStorageSync('random'))
;
  const sessionKey = Taro.getStorageSync('sessionKey');
  sendBle(
    +snApp + 1,
    +snDev,
    config.FUN_SENDER.FUN_RECEIVE_TIME1_REQ,
    newTime,
    newTime.length,
    config.SecurityFlag.FLAG_5_KEY5,
    sessionKey,
    random,
  );
  Taro.setStorageSync('sn_app', +snApp + 1);
};
```

至此，小程序与蓝牙门锁的蓝牙配对建立完成。

后续通信

后续的数据通信，采用 sessionKey 和 random 加密，具体指令及数据参考下文的功能码描述。

发送数据示例代码 (DP 下发)：

```
const random = Taro.getStorageSync('random');
const sn = Taro.getStorageSync('sn_app');
const dps = getDpDemo();
sendBle(
  +sn + 1,
  0,
  config.FUN_SENDER.FUN_SENDER_DPS,
  dps,
  dps.length,
  config.SecurityFlag.FLAG_5_KEY5,
  Taro.getStorageSync('sessionKey'),
  util.HexString2Bytes(random),
);
Taro.setStorageSync('sn_app', +sn + 1);
const getDpDemo = () => {
  const dpCode = 27;
  const dpType = 4;
  const dpValue = 2;
  const dpValueByte = dpDataEncode(dpType, dpValue);
  return [dpCode, dpType, dpValueByte.length, ...dpValueByte];
};
```

接收数据示例代码（DP 上报)：

```
const dataReducer = (sn, sn_ack, code, data) => {
  switch (code) {
    case config.FUN_SENDER.FUN_RECEIVE_DP:
      const dps1 = parseData.dpsDataParse(data);
      response();
      report(dps1);
      // ...
      break;
    case config.FUN_SENDER.FUN_RECEIVE_TIME_DP:
      const dps2 = parseData.dpsDataWithTimestampParse(data);
      response();
      report(dps2);
      // ...
      break;
    default:
      console.log('暂不支持的指令', code);
      break;
  }
};
const response = () => {
  // 发送响应包
};
const report = () = {
  // 接收到蓝牙门锁的dp上报后，需再次将该dp信息上报到云端
}
```

特殊的开门命令

考虑到安全性问题，开门命令需从云端获取数据后下发

```
const sn = Taro.getStorageSync('sn_app');
const random = Taro.getStorageSync('random');
let res = await unlockInstruction({
  device_id: this.data.id,
  bluetooth_sn: +sn + 1,
  device_random: random,
});
if (!res) return { success: false, msg: '开门数据获取失败' };
const { next_sn, instruction } = res;
res = sendCommandBle(util.stringToByte(util.hexToString(instruction)
));
Taro.setStorageSync('sn_app', next_sn);
```

5 功能码描述

```
import { config } from '@tuya-wx/ble-sdk';
const { FUN_SENDER } = config; // 蓝牙命令集合
```

命令帧和响应帧具有相同的功能码，响应帧的 ack_sn 值就是命令帧的 sn

5.1 查询设备信息（0x0000）

```
FUN_SENDER.FUN_SENDER_DEVICE_INFO;
```

命令帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_DEVI
data_len	2	数据长度	0x0000
data	0	数据	null

示例代码：

```
sendBle(
  23, // sn
  0, // ack_sn
  FUN_SENDER.FUN_SENDER_DEVICE_INFO, //code
  null, // data
  0, // data_len
  SecurityFlag.FLAG_4_KEY4,
  secretKey4,
  random,
);
```

响应帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_DEVI
data_len	2	数据长度	0x0053
data	83	数据	设备信息

示例代码：

```
import { parseData } from '@tuya-wx/ble-sdk';
const dataParsed = parseData.parseSearchDeviceBData(data);
```

5.2 发起配对请求（0x0001）

```
FUN_SENDER.FUN_SENDER_PAIR;
```

命令帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_PAIR
data_len	2	数据长度	0x002c
data	44	数据	见下文

data 数据格式

字节数	1-16	17-22	23-44
数据内容	Device_id	Login_key	Device_virtual_id

示例代码：

```
sendBle(
  23, // sn
  0, // ack_sn
  FUN_SENDER.FUN_SENDER_PAIR, //code
  [...Device_id, ...Login_key, ...Device_virtual_id], // data
  0x002c, // data_len
  SecurityFlag.FLAG_5_KEY5,
  sessionKey,
  random,
);
```

响应帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn

字段	长度	说明	值
code	2	功能码	FUN_SENDER.FUN_SENDER_PAIR
data_len	2	数据长度	0x0001
data	1	数据	状态码：0-成功；1-失败；2-已经在绑定状态

5.3 命令下发（0x0002）

```
1 FUN_SENDER.FUN_SENDER_DPS;
```

命令帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_DPS
data_len	2	数据长度	len
data	len	数据	见下文

data 数据格式（DP 数据）：

字节数	1	2	3	4 ～3+Dp_len
数据内容	Dp_id	Dp_type	Dp_len	Dp_data

- Dp_id：在开发平台注册的 dp_id 序号。

- Dp_type：1 个字节。

DP 类型	DT_RAW	DT_BOOL	DT_VALUE	DT_STRING	DT_ENUM	DT_BITMAP
Dp_type	0	1	2	3	4	5

- Dp_len:，数据长度，最大 255
- Dp_data：数据，dp_len 个字节。

示例代码：

```
sendBle(
  23, // sn
  0, // ack_sn
  FUN_SENDER.FUN_SENDER_DPS, //code
  [6, 1, 1, 0], // data
  4, // data_len
  SecurityFlag.FLAG_5_KEY5,
  sessionKey,
  random,
);
```

响应帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_DPS(
data_len	2	数据长度	0x0001
data	1	数据	状态码：0-成功；其他-失败

5.4 设备解绑（0x0005）

```
FUN_SENDER.FUN_SENDER_UNBIND;
```

命令帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_UNBI
data_len	2	数据长度	0x0000
data	0	数据	null

示例代码：

```
sendBle(
  23, // sn
  0, // ack_sn
  FUN_SENDER.FUN_SENDER_UNBIND, //code
  null, // data
  0, // data_len
  SecurityFlag.FLAG_5_KEY5,
  sessionKey,
  random,
);
```

响应帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn

字段	长度	说明	值
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_UNBI
data_len	2	数据长度	0x0001
data	1	数据	状态码：0-成功；其他-失败

5.5 设备重置（0x0006）

```
1 FUN_SENDER.FUN_SENDER_DEVICE_RESET;
```

命令帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_DEVI
data_len	2	数据长度	0x0000
data	0	数据	null

示例代码：

```
sendBle(
  23, // sn
  0, // ack_sn
  FUN_SENDER.FUN_SENDER_DEVICE_RESET, //code
  null, // data
  0, // data_len
  SecurityFlag.FLAG_5_KEY5,
  sessionKey,
  random,
);
```

响应帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_DEVI
data_len	2	数据长度	0x0001
data	1	数据	状态码：0-成功；其他-失败

5.6 状态上报（0x8001）

```
FUN_SENDER.FUN_RECEIVE_DP;
```

命令帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_RECEIVE_DP(0

字段	长度	说明	值
data_len	2	数据长度	len
data	len	数据	见下文

data 数据格式（DP 数据）：

DP 数组	dp-1				～	dp-n			
字节数	1	2	3	4 ～ 3+Dp_	～	n	n+1	n+2	n+3 ～ n+2+Dp_len
数据内容	Dp_id	Dp_type	Dp_len	Dp_data	～	Dp_id	Dp_type	Dp_len	Dp_data

- Dp_id：在开发平台注册的 dp_id 序号。
- Dp_type：1 个字节。

DP 类型	DT_RAW	DT_BOOL	DT_VALUE	DT_STRING	DT_ENUM	DT_BITMAP
Dp_type	0	1	2	3	4	5

- Dp_len:，数据长度，最大 255
- Dp_data：数据，dp_len 个字节。

注：小程序接收到 DP 上报后，需要通过云函数接口，将 DP 信息上报到云端记录。

示例代码：

```
import { parseData } from '@tuya-wx/ble-sdk';
const dps = parseData.dpsDataParse(data);
```

响应帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_SENDER_DEVI
data_len	2	数据长度	0x0001
data	1	数据	状态码：0-成功；其他-失败

示例代码：

```
sendBle(
  1, // sn
  23, // ack_sn
  FUN_SENDER.FUN_RECEIVE_DP, //code
  [0], // data
  1, // data_len
  SecurityFlag.FLAG_5_KEY5,
  sessionKey,
  random,
);
```

5.7 带时间戳状态数据上报（0x8003）

```
FUN_SENDER.FUN_RECEIVE_TIME_DP;
```

命令帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn

字段	长度	说明	值
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_RECEIVE_TIME
data_len	2	数据长度	len
data	len	数据	见下文

- data 数据格式（DP 数据）：

数据单元	时间戳		dp-1				～	dp-n			
字节数	1	n	1	2	3	4～3+Dp_	～	n	n+1	n+2	n+3～n+2+Dp_len
数据内容	timeType	time	Dp_id	Dp_type	Dp_len	Dp_data	～	Dp_id	Dp_type	Dp_len	Dp_data

- TimeType：
  0 表示后面传输的数据是 13 字节 ms 级时间字符串，例如”1553932355000”，13 字节 ms 级，2019/3/30 15:52:35
  1 表示后面 time 是 4 字节 unix 秒级时间戳，大端格式传输。
- Dp_id：
  在开发平台注册的 dp_id 序号。
- Dp_type：
  1 个字节。

DP 类型	DT_RAW	DT_BOOL	DT_VALUE	DT_STRING	DT_ENUM	DT_BITMAP
Dp_type	0	1	2	3	4	5

- Dp_len:，数据长度，最大 255
- Dp_data：数据，dp_len 个字节。

示例代码：

```
import { parseData } from '@tuya-wx/ble-sdk';
const dps = parseData.dpsDataWithTimestampParse(data);
```

响应帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_RECEIVE_TIME
data_len	2	数据长度	0x0001
data	1	数据	状态码：0-成功；其他-失败

示例代码：

```
sendBle(
  1, // sn
  23, // ack_sn
  FUN_SENDER.FUN_RECEIVE_TIME_DP, //code
  [0], // data
  1, // data_len
  SecurityFlag.FLAG_5_KEY5,
  sessionKey,
  random,
);
```

5.8 设备获取实时时间 1（0x8011）

```
FUN_SENDER.FUN_RECEIVE_TIME1_REQ;
```

命令帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_RECEIVE_TIME
data_len	2	数据长度	0x00
data	0	数据	null

响应帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_RECEIVE_TIME
data_len	2	数据长度	0x000f
data	15	数据	见下文

- data 数据格式：

13 字节字符串	2 字节时区（有符号型）
Unix ms 级时间	Time_zone：实际时区的 100 倍，例如北京东八区为 8x100=800，西 7.5 区为-750；

示例代码：

```
// timestamp 为从云函数'timer.currentTime'获取的时间戳
const data = timestamp
  .toString()
  .split('')
  .map(num => num.charCodeAt());
data.push(3, 20); // 800 = 0x0320
sendBle(
  1, // sn
  23, // ack_sn
  FUN_SENDER.FUN_RECEIVE_TIME1_REQ, //code
  data, // data
  15, // data_len
  SecurityFlag.FLAG_5_KEY5,
  sessionKey,
  random,
);
```

5.9 设备获取实时时间 2（0x8012）

```
FUN_SENDER.FUN_RECEIVE_TIME2_REQ;
```

命令帧：BLE 门锁 -> 小程序

字段	长度	说明	值
sn	4	请求包序列号	sn
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_RECEIVE_TIME
data_len	2	数据长度	0x00
data	0	数据	null

响应帧：小程序 -> BLE 门锁

字段	长度	说明	值
sn	4	请求包序列号	sn

字段	长度	说明	值
ack_sn	4	响应包序列号	ack_sn
code	2	功能码	FUN_SENDER.FUN_RECEIVE_TIME
data_len	2	数据长度	0x0009
data	9	数据	见下文

- data 数据格式：

字节数	1	2	3	4	5	6	7	8	9
数据内容	年	月	日	时	分	秒	星期	时区(高位)	时区(低位)

注：
年：取后两位，如 2019 年，只取’19’，即为 0x13
星期：0-6，0 代表星期日。
时区：实际时区的 100 倍，例如北京东八区为 8x100=800，西 7.5 区为-750；
如’北京时间 2019 年 4 月 28 日 12:23:25 星期二’，应按如下顺序发送：0x13,0x04,0x1C,0x0C,0x17,0x19,0x02,0x03,0x20。
注：如果 app 从云端获取不到时间，app 需要回复全 0 数据。

示例代码：

```
sendBle(
  1, // sn
  23, // ack_sn
  FUN_SENDER.FUN_RECEIVE_TIME2_REQ, //code
  [19, 4, 28, 12, 23, 25, 2, 3, 20], // data
  9, // data_len
  SecurityFlag.FLAG_5_KEY5,
  sessionKey,
  random,
);
```

6 云函数接口文档

6.1 获取设备详情

接口地址

```
action :device.details
```

请求参数

参数名	类型	说明	必填
device_id	String	设备 Id	是

请求示例

```
{
  "action": "device.details",
  "params": {
    "device_id": "vdevo160230162935706"
  }
}
```

返回参数

参数名	类型	说明
code	Integer	错误响应码，成功时为空（详情见见错误码）
success	Boolean	是否成功：（true：成功，false：失败）
t	Long	响应时间
msg	String	请求失败的信息，成功为空

参数名	类型	说明
result	Object	设备详情

result 说明

参数名	类型	说明
local_key	String	密钥
uid	String	用户 ID
uuid	String	设备唯一标识

请求成功返回示例

```
{
  "result": {
    "local_key": "525e9d7a90c39411",
    "uuid": "c3ac4b0d54af5703",
    "uid": "ay16009313532665EC8Y"
  },
  "success": true,
  "t": 1603679935012
}
```

请求失败返回示例

```
{
  "success": false,
  "code": 500, // 错误码，详细请见错误码文档
  "msg": "system error,please contact the admin"
}
```

6.2 配对密钥

接口地址

```
action :device.pairingAuthKey
```

请求参数

参数名	类型	说明	必填
home_id	Long	家庭 Id	是
uuid	String	设备唯一标识	是
uid	String	用户 ID	是

请求示例

```
{
  "action": "device.pairingAuthKey",
  "params": {
    "home_id": "25761214",
    "uuid": "c3ac4b0d54af5703",
    "uid": "ay16009313532665EC8Y"
  }
}
```

返回参数

参数名	类型	说明
code	Integer	错误响应码，成功时为空（详情见见错误码）
success	Boolean	是否成功：（true：成功，false：失败）

参数名	类型	说明
t	Long	响应时间
msg	String	请求失败的信息，成功为空
result	Object	配网密钥

result 说明

参数名	类型	说明
encrypted_auth_key	String	密钥 1
random	String	登入密钥

请求成功返回示例

```
{
  "result": {
    "encrypted_auth_key": "456789056789",
    "rondom": "6783456789"
  },
  "success": true,
  "t": 1603679935012
}
```

请求失败返回示例

```
{
  "success": false,
  "code": 500, // 错误码，详细请见错误码文档
  "msg": "system error,please contact the admin"
}
```

6.3 服务器时间

接口地址

```
action :timer.currentTime
```

请求参数

无

请求示例

```
{
  "action": "timer.currentTime",
  "params": {}
}
```

返回参数

参数名	类型	说明
code	Integer	错误响应码，成功时为空（详情见见错误码）
success	Boolean	是否成功：（true：成功，false：失败）
t	Long	响应时间
msg	String	请求失败的信息，成功为空
result	Long	服务器时间戳

请求成功返回示例

```
{
  "result": true,
  "success": 1607087405739,
  "t": 1603679935012
}
```

请求失败返回示例

```
{
  "success": false,
  "code": 500, // 错误码，详细请见错误码文档
  "msg": "system error,please contact the admin"
}
```

6.4 设备 DP 状态上报

接口地址

```
action :device.dpReport
```

请求参数

参数名	类型	参数类型	说明	必填
device_id	String	URI	设备 Id	是
dps	List	BODY	DP 状态列表	是
push_mobile	Boolean	BODY	是否推送给 App	是

dps

参数名	类型	参数类型	说明	必填
dp_id	String	BODY	DpId	是
dp_value	Object	BODY	Dp 状态	是

请求示例

```
{
  "action": "device.dpReport",
  "params": {
    "device_id": "vdevo160230162935706",
    "push_mobile": false,
    "dps": [
      {
        "dp_id": "2",
        "dp_value": 100
      }
    ]
  }
}
```

返回参数

参数名	类型	说明
code	Integer	错误响应码，成功时为空（详情见见错误码）
success	Boolean	是否成功：（true：成功，false：失败）
t	Long	响应时间
msg	String	请求失败的信息，成功为空
result	Boolean	是否成功

请求成功返回示例

```
{
  "result": true,
  "success": true,
  "t": 1603679935012
}
```

请求失败返回示例

```
{
  "success": false,
  "code": 500, // 错误码，详细请见错误码文档
  "msg": "system error,please contact the admin"
}
```

6.5 蓝牙近程开门

接口地址

```
acton:   doorLock.unlockInstruction
```

请求参数

参数名	类型	参数类型	说明	必填
device_id	String	URI	设备 Id	是
bluetooth_sn	Integer	BODY	蓝牙通信时防重放 sn	是
device_random	String	BODY	门锁设备随机数	是

请求示例

```
{
  "action": "doorLock.unlockInstruction",
  "params": {
    "device_id": "vdevo153459260090544",
    "bluetooth_sn": 123,
    "device_random": "123"
  }
}
```

返回参数

参数名	类型	说明
code	Integer	错误响应码，成功时为空（详情见见错误码）
success	Boolean	是否成功：（true：成功，false：失败）
t	Long	响应时间
msg	String	请求失败的信息，成功为空
result	Object	开门指令信息

result 说明

参数名	类型	说明
instruction	String	开门指令
next_sn	Integer	下一个蓝牙通信时防重放 sn

请求成功返回示例

```
{
  "result": {
    "next_sn": 123,
    "instruction": "9wxxoLM"
  },
  "success": true,
  "t": 1592899848757
}
```

请求失败返回示例

```
{
  "success": false,
  "code": 500, // 错误码，详细请见错误码文档
  "msg": "system error,please contact the admin"
}
```