一、身份认证以及加解密方式

密钥的产生和体系

产生

数据密钥应具备随机产生特性，密钥产生后要检查密钥的有效性，弱密钥和半弱密钥需被剔除。

运营商加入信息交换时，必须申请独立的密钥文件，密钥可由运营商协商产生。

体系

每个运营商与平台交互前需要分配平台标识 (OperatorID)、平台密钥 (OperatorSecret)、消息密钥（DataSecret）、消息密钥初始化向量（DataSecretIV）和签名密钥（SigSecret）。

1)平台标识（OperatorID）：固定9位（在【企查查】等网站上可以找到），运营商的组织机构代码，作为运营商的唯一标示。

2) 平台密钥 (OperatorSecret):固定16位，可采用32H、48H 和64H，由 0-F 字符组成，为申请认证使用。

3)消息密钥（DataSecret）:固定16位，用于对所有接口中 Data 信息进行加密。

4)消息密钥初始化向量（DataSecretIV）： 固定16位，用户 AES 加密过程的混合加密。

5)签名密钥（SigSecret）：固定16位，由 0-F 字符组成，为签名的加密密钥。

平台认证方式及规则

平台认证方法

平台认证宜采取身份认证和访问控制相结合的方式进行。

身份认证可采取用户名/口令认证、密钥认证或数字证书认证等方式进行；访问控制可采取 IP 访问控制、时间访问控制等多种手段结合。

用户身份认证成功后授予 Token，每次向服务端请求资源的时候需要带着服务端签发的 Token，服务端验证 Token 成功后，才返回请求的数据。Token 的有效期由服务方确定，最长不应超过7天，Token 丢失或失效后需要再次发起认证服务。

数据传输方式及规则

接口调用方式

所有接口均使用 HTTP(S)/POST 方式传输参数,传输过程中应包含消息头和消息主体两部分。

消息头规范

消息头一般需包含内容类型和授权信息（Authorization）。

内容类型（Content-Type）字段用于标识请求中的消息主体的编码方式，本标准中所规范的信息交换内容均采用 JSON 的方式，参数信息采用 utf-8编码，因此需要配置消息头中的 Content-Type 为 application/json;charset=utf-8。

授权信息（Authorization）字段用于证明客户端有权查看某个资源，本标准中所规范的授权信息采用凭证（Token）的方式，因此需要在配置消息头中的 Authorization 为 Bearer Token。

post 示例：

请求:

{"Sig":"E93ADAC98D7D203CD47E28409898CD5E","Data":"VndeE6fncXunSVNtWKaduh+b4XsqH5aDtXSUm3RdhQ5O1sBbmTuMnU5xxY7rFdOWoF99bXD5RihmBj3eoplj BA==","OperatorID":"MW5F32R7","TimeStamp":"20220602160830","Seq":"0001"}

返回:

{"ret":0,"msg":"请求成功
","data":"VndeE6fncXunSVNtWKaduhTGXsM0cSrlGM4EA2vpsyYClZQbo4O7hn7+0S
HnqlkbaizYqjPdXQ0zZJ1lnXl+jo3Blgj/30i2DAlK+HiEKbQFuqSwh+wEc9tJiqPOkXLC6
mQQQAovW4ylFqrqsTzYlDrjnGrpxPAucj4+FCTyl4lQoA9Bjl879uXFbxqJUe1lfXZlHpZlP
uN4lmL+4KakybKhsMm8Ut+ZQlkGlQJluMr5qBDgmxjSBWvkBb2xoW7A1nZESpQGltK
QxvdvpRrOfQ==","Sig":"15163CB3D8D950E7E4C4450B2D39A08A"}

请求参数规则

一般由运营商标识（OperatorID）、参数内容（Data）JSON 串、时间戳（TimeStamp）、自增序列（Seq）和数字签名（Sig）组成。

表 1　请求消息主体内容表

参数名	说明	举例	是否必填
OperatorID	运营商标识		
Data	各接口具体加密后的参数信息	8TRL3nAAPQyKFp7a+XaOZ+K+WmpSN2uf9m+ZYZawnwtSGIJdL2Fg2PyZPHCM+3UZUqC43vBdmWSrkFBMUDLm0g==	是
TimeStamp	时间戳	接口请求时时间戳信息,格式为 yyyyMMddHHmmss	是
Seq	自增序列	4 位自增序列取自时间戳，同一秒内按序列自增长，新秒重计。如 0001	是
Sig	参数签名	加签参数为 OperatorID+Data+TimeStamp+Seq	是

返回参数规则

数据传输接口的返回参数一般由返回值（Ret）、返回信息（Msg）、参数内容（Data）和数字签名（Sig）组成。

1)Ret:必填字段，返回编码参考下表。

2)Msg:必填字段，有错误表示具体错误信息，无错误返回成功信息。

3)Data:必填字段，参数内容,采用 utf-8编码，JSON 格式。

表 2　返回消息主体内容表

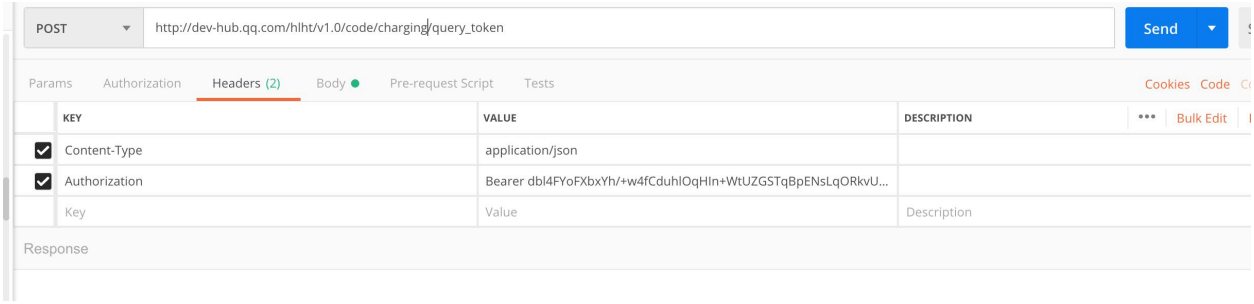
参数名	说明	举例	是否必填
Ret	返回参数编码		是
Msg	返回消息字符串		是
Data	业务加密数据	hrJar2kPUP7p1JvXmPvyJ7hBe21jHG+j6x8hWxFNG6+Z4WxYRhKtSb7wybOixfacVc3aDriNAvwlQjJS91LNzCOgSI8Hb57uPQ5pFmXcHvJIZP5qvxqA2HDXQVyJEovV5Lfoe7i/JdCYu8L3Gh/2AirwvFK2KQCrfYgR8phCws4q1Omc6RRnKQJdYchpi6pctewccNEdQB2sh0Yxm6imMxlCEsa0J6NZJdxBunpyNoY=	是
Sig	签名	Ret+Msg+Data	是

表 3　返回参数编码表

Ret 值	说明
-1	系统繁忙，此时请求方稍后重试
0	请求成功
4001	签名错误
4002	Token 错误
4003	POST 参数不合法,缺少必须的示例：OperatorID,sig,TimeStamp,Data，Seq 五个参数
4004	请求的业务参数不合法，各接口定义自己的必须参数
500	系统错误

post 示例：

Token：



请求：

{"Data":"MGNFfGV3Ooqc+Uqd9KmDmKhBbKtRwFjRQYGkIXxTzsnE4QMzye3WW05zTBTiK7iLs76b5eXYrt2zZFO6R/YMR/Zbk91R26dkmaoUiWviv1kmtQqgXhxNJ8i6QZS5oAnCk5lf90AwNStVz7fwL3y6H+h5Tj55lGCVVF2Pr1BzGEw=","OperatorID":"123456789","TimeStamp":"1654156380","Sig":"DD0BA4BA4D122BE9EE8D51F708E45C60","Seq":"594"}

返回：

{"Data":"J3OPNG7s6nVbKeCHQVDs0g==","Msg":"请求成功
","Ret":0,"Sig":"15163CB3D8D950E7E4C4450B2D39A08A"}

批量数据传输

数据传输接口中的 Data 字段可为数组型的 JSON 格式，数据发送方可通过该字段实现批量数据的传输。

数据加解密规则

消息发送方需要对 Data 字段中涉及交易及隐私等数据利用消息密钥（DataSecret）进行加密，加密算法宜使用 AES 128位加密，加密模式采用 CBC，填充模式采用 PKCS5Padding 方式。

消息接收方收到消息之后，根据消息密钥（DataSecret）对消息体中的 Data 数据进行解密，校验参数合法性等后续业务处理。

参数签名规范

参数签名要求

参数签名采用 HMAC-MD5算法，采用 MD5 作为散列函数，通过签名密钥（SigSecret）对整个消息主体进行加密，然后采用 Md5信息摘要的方式形成新的密文，参数签名要求大写。

请求参数的签名顺序按照消息体顺序拼接后执行，拼接顺序为运营商标识（OperatorID）、参数内容（Data）、时间戳（TimeStamp）、自增序列（Seq）。

返回参数的签名顺序按照消息体顺序拼接后执行，拼接顺序为返回值（Ret）、返回信息（Msg）、参数内容（Data）。

二、接口（信息交互均是 json 形式）

认证接口（塑云调用道闸系统）

此接口用于平台之间认证 Token 的申请，Token 作为全局唯一凭证，调用各接口时均需要使用。此接口也应实现加解密规范要求和验签要求。Token 的有效时间一般为7天。

接口定义

接口名称：query_token
接口使用方法：由服务端实现此接口，需求端调用。

输入参数

参数名称	定义	参数类型	描述
运营商标识	OperatorID	字符串	运营商组织机构代码
运营商密钥	OperatorSecret	字符串	运营商分配的唯一识别密钥

返回值

参数名称	定义	参数类型	描述
运营商标识	OperatorID	字符串	运营商组织机构代码
成功状态	SuccStat	整型	0:成功； 1:失败
获取的凭证	AccessToken	字符串	全局唯一凭证
凭证有效期	TokenAvailableTime	整型	凭证有效期，单位秒
失败原因	FailReason	整型	0:无； 1:无此运营商； 2:密钥错误； 3～99:自定义

道闸接口（塑云调用道闸系统）

接口定义

方法名称：chargeorder

输入参数

字段	类型	含义	字段限制说明
parkId	String	停车场站 id	必填 0-100 字符
orderNo	String	订单号	必填 0-100 字符
plateNo	String	车牌号(道闸方提供)	必填 0-100 字符
startTime	String	充电开始时间	yyyy-MM-dd HH:mm:ss
endTime	String	充电结束时间	yyyy-MM-dd HH:mm:ss
saleValue	String	减免时长，单位为分钟	作为真正减免依据

返回值

参数名称	定义	参数类型	描述
状态	SuccStat	整型	0:成功; 1:失败
原因	FailReason	字符串	成功或失败原因

三、代码示例（java）

加密

```
public static String encrypt (String data, String dataSecretIV, String dataSecret) {
  try {
    if (data == null || data.isEmpty()) {
      return "";
    }
    IvParameterSpec zeroIv = new IvParameterSpec(dataSecretIV.getBytes());
    SecretKeySpec key = new SecretKeySpec(dataSecret.getBytes(), "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

```
cipher.init(Cipher.ENCRYPT_MODE, key, zeroIv);
      byte[] encryptedData = cipher.doFinal(data.getBytes( "UTF-8"));

      return Base64.encode(encryptedData).replace("\n", "").replace("\r", "");
    } catch (Exception e) {
      log.error("AesUtil-encrypt-error: param:
encrypted:{},dataSecretIV:{},dataSecret:{}",data,dataSecretIV,dataSecret,e);
      return "";
    }
}
```

解密

```
public static String decrypt (String data, String dataSecretIV, String dataSecret) {
    try {
      if (data == null || data.isEmpty()) {
        return "";
      }
      byte[] byteMi =Base64.decode(data);
      IvParameterSpec zeroIv = new IvParameterSpec(dataSecretIV.getBytes());
      SecretKeySpec key = new SecretKeySpec(dataSecret.getBytes(), "AES");
      Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      cipher.init(Cipher.DECRYPT_MODE, key, zeroIv);
      byte[] decryptedData = cipher.doFinal(byteMi);
      return new String(decryptedData, "UTF-8");
    } catch (Exception e) {
      return "";
    }
}
```

签名

```
public static String hmacSign (String aValue, String sigSecret) {
    byte[] k_ipad = new byte[64];
```

```
    byte[] k_opad = new byte[64];
    byte[] keyb;
    byte[] value;
    keyb = sigSecret.getBytes(StandardCharsets.UTF_8);
    value = aValue.getBytes(StandardCharsets.UTF_8);

    Arrays.fill(k_ipad, keyb.length, 64, (byte) 54);
    Arrays.fill(k_opad, keyb.length, 64, (byte) 92);
    for (int i = 0; i < keyb.length; i++) {
        k_ipad[i] = (byte) (keyb[i] ^ 0x36);
        k_opad[i] = (byte) (keyb[i] ^ 0x5c);
    }

    MessageDigest md;
    try {
        md = MessageDigest.getInstance("MD5");
    } catch (NoSuchAlgorithmException e) {
        return null;
    }
    md.update(k_ipad);
    md.update(value);
    byte[] dg = md.digest();
    md.reset();
    md.update(k_opad);
    md.update(dg, 0, 16);
    dg = md.digest();
    return toHex(dg);
}
```

转成 16 进制

```
public static String toHex (byte[] input) {
    if (input == null) {
        return null;
    }
    StringBuffer output = new StringBuffer(input.length * 2);
```

```
    for (byte b : input) {
        int current = b & 0xff;
        if (current < 16) {
            output.append("0");
        }
        output.append(Integer.toString(current, 16));
    }
    return output.toString().toUpperCase();
}
```

加解密、签名、token 调用

加密数据：*encryptD*ata=*encrypt*(*decryptD*ata, DataSecretIv, DataSecret);
*解密数据：decryptD*ata=*decrypt*(*encryptD*ata, DataSecretIv, DataSecret)
签名数据：
入参签名方式：

sig = *hmacSign*(OperatorID+ Data+ TimeStamp+Seq, SigSecret)
出参签名方式：
sig = *hmacSign*(ret + msg + data, SigSecret)
token 生成：token=encrypt(OperatorID+uuid, DataSecretIV, OperatorSecret)