包名：com.bw30.zsch

强制更新

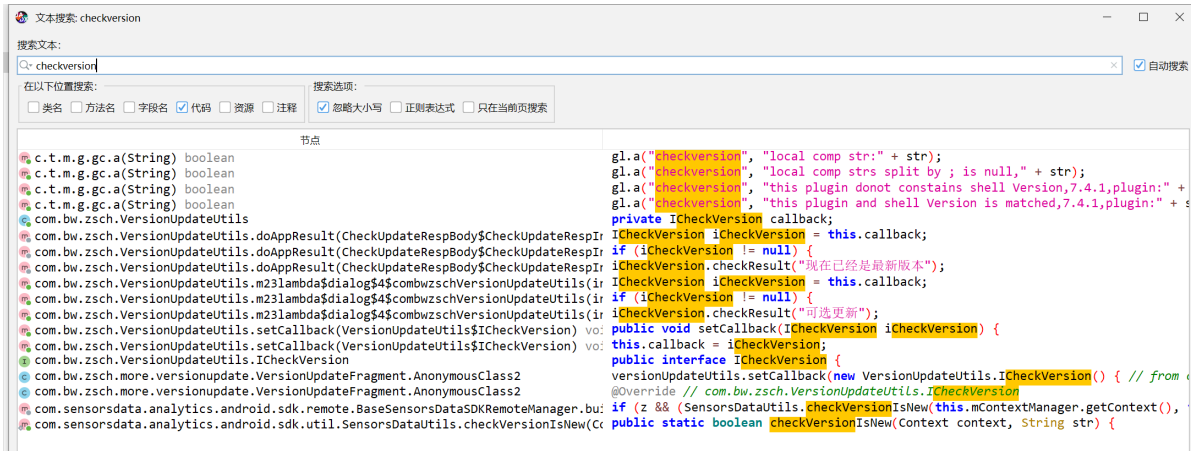
使用 jadx 反编译 App 时，发现应用采用了 360 加固。首先尝试使用 `frida-dump` 进行脱壳，但未成功，脱下来的dex不完整。随后借助一个免费的脱壳网站完成脱壳，脱壳后的文件看起来较为完整。

现在打开app后发现有一个强制更新，我们去找一下它的代码，看看能不能hook掉：

jadx反编译后直接搜索关键字(立即更新、发现新版本、版本等)没搜到，我们去抓一下包看看！

Charles直接搜索关键字也没有，看了一眼请求体和响应体是加密的

再去jadx中搜一下经典的检查版本的关键字：`check、version`吧



有一些，我们逐一看看!

```
529     protected void doAppResult(CheckUpdateRespBody.CheckUpdateRespInfo checkUpdateRespInfo) {
            int i;
            LogUtil.i(checkUpdateRespInfo.getType().getName() + "");
```

看这个参数的名字，`checkUpdateRespInfo`看着像关于检查更新请求的

```java
protected void doAppResult(CheckUpdateRespBody.CheckUpdateRespInfo checkUpdateRespInfo) {
    int i;
    LogUtil.i(checkUpdateRespInfo.getType().getName() + "");
    UpdateStatus updateStatus = checkUpdateRespInfo.getUpdateStatus();
    if (updateStatus == UpdateStatus.NOT_NEED_UPDATE) {
        ICheckVersion iCheckVersion = this.callback;
        if (iCheckVersion != null) {
            iCheckVersion.checkResult("现在已经是最新版本");
        }
    } else if (updateStatus == UpdateStatus.FORCE_UPDATE) {
        try {
            int i2 = numFirst;
            if (i2 == 0) {
                numFirst = i2 + 1;
                new UpdateDialog(activity, checkUpdateRespInfo.getUrl(), checkUpdateRespInfo.getTip(), 1, checkUpd
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else if (updateStatus == UpdateStatus.SELECT_UPDATE && (i = numFirst) == 0) {
        numFirst = i + 1;
        if (NotificationDownloadManager.isDownLoading) {
            if (NotificationDownloadManager.downState == 1) {
                activity.sendBroadcast(new Intent(activity, DownBrcastReceiver.class));
                DialogUtils.showToast(activity, "新版本继续下载中", 1);
                return;
            }
            DialogUtils.showToast(activity, "新版本正在下载中", 1);
            return;
        }
```

可以看到在`doAppResult`方法里有三个if判断，分别是判断`updateStatus`的值，是否等于`UpdateStatus.NOT_NEED_UPDATE`、`UpdateStatus.FORCE_UPDATE`、`UpdateStatus.SELECT_UPDATE`

我们点这三个值进去可以看到这是一个枚举类，里边有三个枚举变量，分别代表`不需要更新`、`可选更新`、`强制更新`。

```java
/* Loaded from: D:\桌面\资料\带带弟弟学爬虫
8 public enum UpdateStatus {
    NOT_NEED_UPDATE(1, "不需要更新"),
    SELECT_UPDATE(2, "可选更新"),
    FORCE_UPDATE(3, "强制更新");

    private int code;
```

这里是强制更新，所以应该是返回3：`FORCE_UPDATE`。

可以看到这个`updateStatus`是通过调用`checkUpdateRespInfo.getUpdateStatus()`得到的

```
UpdateStatus updateStatus = checkUpdateRespInfo.getUpdateStatus();
```

所以我们可以直接去hook这个`checkUpdateRespInfo.getUpdateStatus()`，让他的返回值为：`NOT_NEED_UPDATE`或者`UpdateStatus.SELECT_UPDATE`，这样就直接不需要更新就进入app了

```
public UpdateStatus getUpdateStatus() {
        return this.updateStatus;
    }
```

我们直接去使用spwan的方式去hook，发现hook不到，找不到对应的类，这应该跟加固有关系，加固后的类加载机制可能有些变化，所以我采用了attach的方式，在软件启动的那一刻，注入。侥幸可以注入，等有时间了需要去看看加壳脱壳，学习学习类加载机制。

这里看到枚举类中，2代表的是选择更新，为了验证返回值是否更换成功，所以我直接让返回值为2。

但是呢，不能直接返回2，直接返回2会报错，但是不影响过掉强制更新。

经过gpt验证，可能是以下原因：

一、核心原因定位

错误信息 `expected return value compatible with...UpdateStatus` 表明：

1. Hook方法返回的枚举实例与目标方法声明的枚举类**不属于同一个ClassLoader加载**
2. 返回对象可能来自其他包名/类名的同名枚举（如子类或动态加载类）
3. Frida-Java-Bridge的类型校验机制触发了兼容性检查失败 [1]

分析完了，是因为返回值类型不匹配。经过豆包提醒，返回值用`valueOf`包裹了一下：

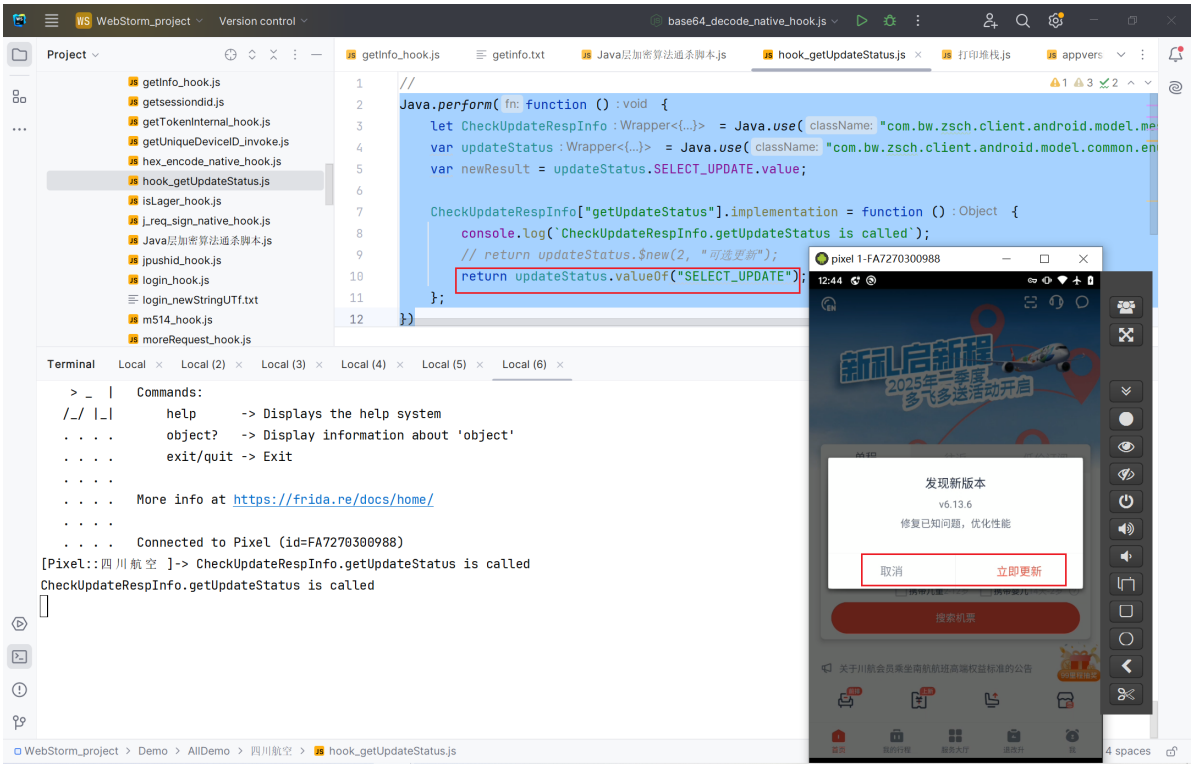
```
Java.perform(function () {
    let CheckUpdateRespInfo =
Java.use("com.bw.zsch.client.android.model.message.CheckUpdateRespBody$CheckUpdateRespInfo");


    //UpdateStatus枚举类
    var updateStatus =
Java.use("com.bw.zsch.client.android.model.common.enumeration.UpdateStatus");

    //hook getUpdateStatus
    CheckUpdateRespInfo["getUpdateStatus"].implementation = function () {
        console.log(`CheckUpdateRespInfo.getUpdateStatus is called`);

        //直接返回SELECT_UPDATE：可选更新
        return updateStatus.valueOf("SELECT_UPDATE");
    };
})
```

效果如下：



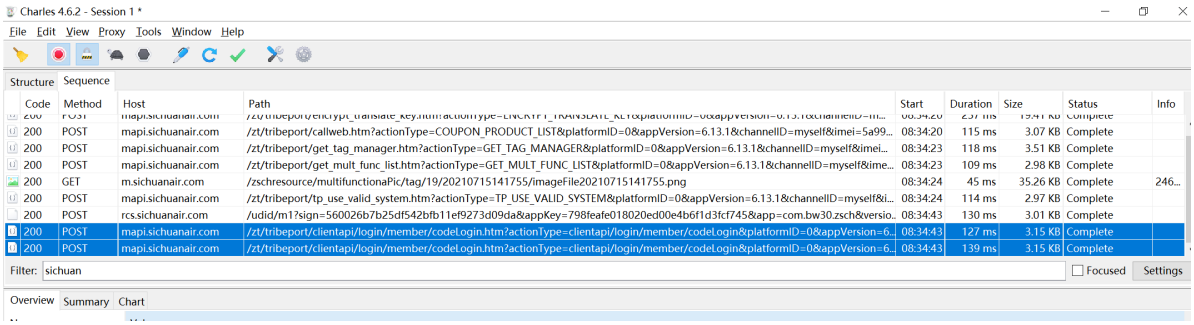
当然，也可以采取离线进入app，然后再联网，也是可以跳过的

登录

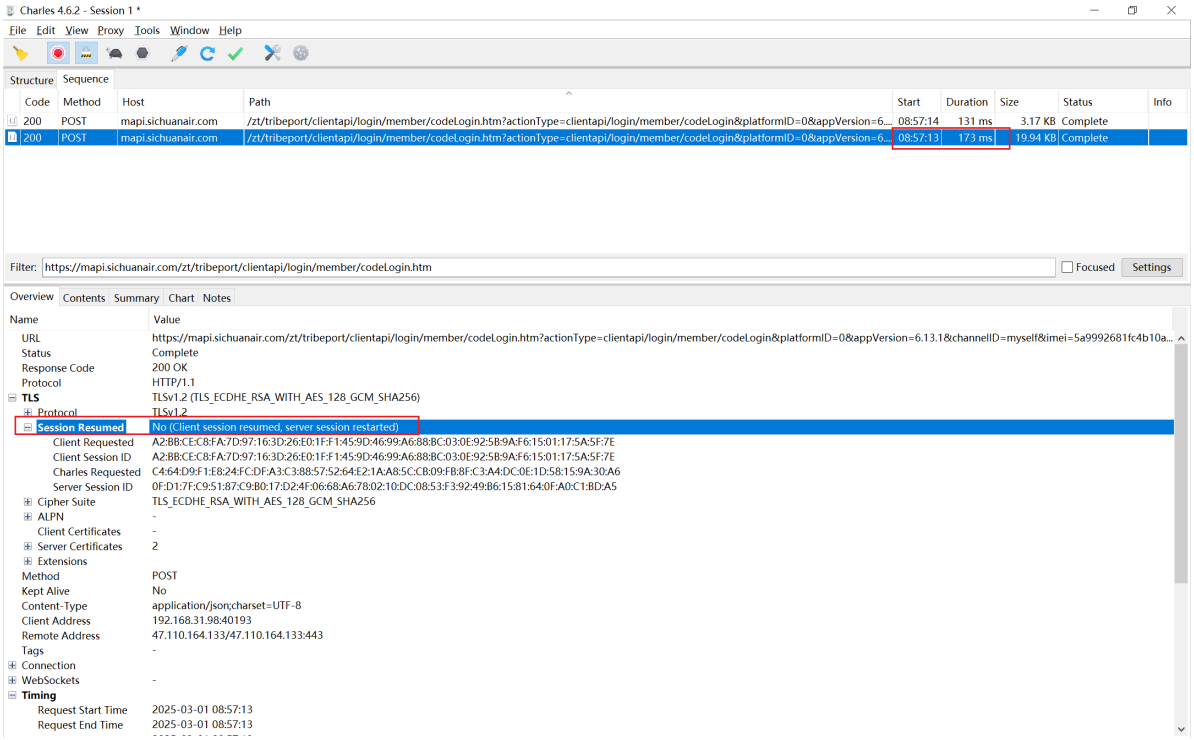
现在看看登录，先抓包！

我用的验证码登录，返回的是验证码错误，但是直接搜索 `验证码错误` 搜不到，因为这个请求体和响应体是加密的了。

这里通过逐一查看请求网址，找到两条高度疑似的：`codeLogin`



但是不知道为什么同一时间有两条？ 是我点到了吗？ 我们再去请求一下看看：



又抓了一次，是有两条登录信息，其中有一条要早1秒，这个早一秒的在`Session Resumed`这里是no，而晚一秒的是yes

在 Charles 抓 HTTPS 包时出现这种情况，原因如下：

“Session Resumed” 及 “No” 的情况

当第一次点击登录发送请求时，“Session Resumed” 表示尝试恢复之前的会话连接来传输数据。而 “No (Client session resumed, server session restarted)” 说明客户端这边成功恢复了会话，但服务器端由于某些原因（比如服务器端的会话管理策略，超时、服务器重启、或者内部的会话刷新机制等）重新启动了会话。这就导致虽然客户端想基于之前的会话来通信，可服务器并不认同这个旧会话，而是开启了新的会话处理这次请求。

“Session Resumed” 及 “Yes” 的情况

第二次请求出现 “Session Resumed” 且是 “Yes”，意味着客户端和服务器之间成功恢复了会话连接。可能是因为第一次请求时服务器重新启动会话后，在第二次请求到达时，服务器识别到客户端的会话相关信息符合预期，允许基于已有的会话继续通信，这样可以减少一些连接建立和初始化的开销，提高数据传输效率。

所以我们去看`Session Resumed`为yse的那个请求

```
请求网址：
https://mapi.sichuanair.com/zt/tribeport/clientapi/login/member/codeLogin.htm
请求方式：POST
请求头：
    Cookie              JSESSIONID=838DF1059ECB8D42CA751AA6A561049F
    Content-Length      1100
    Content-Type        text/plain; charset=ISO-8859-1
    Host                mapi.sichuanair.com
    Connection          Keep-Alive
    User-Agent          Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
    Accept-Encoding     gzip
```

```
携带参数:
    actionType          clientapi/login/member/codeLogin
    platformID          0
    appVersion          6.13.1
    channelID           myself
    imei                5a9992681fc4b10a
    networkOperator
    mac                 02:00:00:00:00:00
    transActionId       517cf7c2-41f2-4705-a458-fe6048fbf9e9
请求体:
GKu4Leqtpn/v2wIlHP7J+WQStqM9H5DMRM11jRMwXSoDwsCOCKzI+Ou5DqukrBkcv4fUxk+hjZN55yS8
WqBpC986aG0sbcse7/r7oMIrvNJN+/I8OgIgTIzn1iRBlnN3L0NbV/1EGBX98tkbNCy6chSKPMNIMBr0
ulJH0S9xv5siLhNJH4G7+R3C/gA1lD+wGqnRE4I0HwAy/dZMVO4oS0PcVI41mRfGcDV3JihapHF6RiR5
eQnb+4sXaeKlbxJrZKQDHqA0nekjvdZZp0NBk4+95/2vmRKhP1/JjsX6XjCzJMDbiymZlJrSPboU1Agd
B5MtYmlXn0uhE9nfD8hMUychjPt6eMhcgba0iUC1gybbXjto1J+EmAncAleA9Yasp4IcKPIxA+zs83/N
/gT2x9hn5ZixJkdKYXqVuCTU+WBzG85sDQfsQg8LKtaR/1QeWFJfW3yvOguPz9ow+m1yDUEVWaLtIIPm
Jupjz8U91e9fKGHI5UUHektIZPGan9fanPj6rNBPs72lY/K+WGlyqA9JrxJKhGWf+fG93CuOxY5w6kuw
5r0lx67AADJGWyA0mXh26zTrLVSHt6OSBn+QMcN9ZXdAtltmO/2Qk+05c3T7lJZDYWepieRQFoHsnN4/
EYOnz7l3Qit46tnCXBp2LcLag1slcF3o3EV4/irKNL3ktOGifdJSz5lBVZVbGsG6Cb8II8uISbpvX6gB
vDSy/8nSH8ZQJw42f3IZMa/t6DplishOZEW97uo22W7ojjy7WiM154QAHcdSFMyLcRfoUN0nLpOpZ8y+
Tb7S3QqDbnbkaFVSm9XXXM6KxOg/3zOuQkF+ODdzwwpvJPRjeULI6sozSpEaAnYooCpcnCWgV3TyZeP4
+YbSbM6kM1R3jzfAcMZBuJvcbcOuiLI6EIXBUKa5M2GoWC9AEQ57pauDOoEH2DgcjFLaT7w/KcneDLVG
IMJ53IccfUn5aAEkUAfL+W2RslUAGJmFo1xbibsPqHbgIXtiHp+b6zpxTtX5xHj4jzpw9W5RvlIfrqg5
YFrJA5DYUs0AOqxAezjeeW1bJcuwOKVgc8TXkuh3vrJCbEiUhUKQl2JUhjM=
```

这里我们先搜索一下请求网址看看：

找到了三条，看类名应该是关于登录的`Activity`，这里要注意到，虽然是三条登录的信息，但是可以看到都是走的`ServiceFactory.getBasciFactory().callWeb`方法

我们现在去hook一下这个`callWeb2(参数1, 参数2, 参数3, 参数4)`，看下它的参数是什么

```
[Pixel::四川航空 ]->
ServiceDataImpl.callWeb2 is called:

obj=CodeLoginReqBody(phone=15512103215,verifyCode=123456,picCode=null,mobileCoun
tryCode=86),

obj2=TPSelfChannelRiskControllerSystemReqBody{validType='0',funcType='null',cons
tId='67c4f6adObm8RHhdjCSVkxHijV49hTBMtpQPWct3',token='null',ip='null',phoneNumbe
r='null',userId='null',userName='null',source='null',extCurrentUrl='null',extCoo
kie='null',extSessionId='null',extAccountExist='null',extPasswordCorrect='null',
extIdState='null',extLoginType='null',extId='null',extIdNo='null',email='null',m
emberLevel=null,serchFlightRequest=null,submitOrderRequest=null,activityRequest=
null,payRequest=null,refundTicketRequest=null},

str=clientapi/login/member/codeLogin,

serviceDataCallback=[object Object]
```

我们现在去看这个 `ServiceFactory.getBasciFactory().callWeb2` 的代码

```
  public void callWeb2(Object obj, Object obj2, String str, final
ServiceDataCallback<ClientApiResult> serviceDataCallback) {
        ClientApiRiskReq clientApiRiskReq = new ClientApiRiskReq();
        clientApiRiskReq.setBody(obj);
        clientApiRiskReq.setRiskReqBody(obj2);
        clientApiRiskReq.setHead(new ReqHead(str))
      //上边四行是封装请求数据的,这里new了一个ClientApiRiskReq的对象，这里很重要，是new的
ClientApiRiskReq对象
      //这里set了三个地方，我们点进ClientApiRiskReq类，可以看到，里边只有一个
RiskReqBody，body和head是继承的父类的

            EncryptUtils2.encrypt(
         //调用方法：EncryptUtils2.encrypt
            参数1：clientApiRiskReq,
          //待加密的请求数据
            参数2：new IEncrypData() {//回调},
            参数3：ClientApiResult.class
        );
    }
```

我们去hook一下这个 `EncryptUtils2.encrypt` 函数

```
[Pixel::四川航空 ]-> EncryptUtils2.encrypt is called:

参数1:
clientApiReq=ClientApiRiskReq{head=ReqHead{proVersion='null',path='null',action=
'clientapi/login/member/codeLogin',transActionId='null',timestamp='null',verify=
'null',sign='null',channelId='null',platformId=null,imie='null',macAddress='null
',uuid='null',idfa='null',jpushId='null',appVersion='null',sessionId='null',toke
nId='null',acounttype='null',callWebAction='null',clientIp='null',checkKey='null
',customerId='null',externalChannel='null'},body=CodeLoginReqBody(phone=15512103
215,verifyCode=123456,picCode=null,mobileCountryCode=86),riskReqBody=TPSelfChann
elRiskControllerSystemReqBody{validType='0',funcType='null',constId='67c4fcean2b
lel4y8QKTT2iSAltlxhjsQxSZZE53',token='null',ip='null',phoneNumber='null',userId=
'null',userName='null',source='null',extCurrentUrl='null',extCookie='null',extSe
ssionId='null',extAccountExist='null',extPasswordCorrect='null',extIdState='null
',extLoginType='null',extId='null',extIdNo='null',email='null',memberLevel=null,
serchFlightRequest=null,submitOrderRequest=null,activityRequest=null,payRequest=
null,refundTicketRequest=null}}ClientApiReq{head=ReqHead{proVersion='null',path=
'null',action='clientapi/login/member/codeLogin',transActionId='null',timestamp=
'null',verify='null',sign='null',channelId='null',platformId=null,imie='null',ma
cAddress='null',uuid='null',idfa='null',jpushId='null',appVersion='null',session
Id='null',tokenId='null',acounttype='null',callWebAction='null',clientIp='null',
checkKey='null',customerId='null',externalChannel='null'},body=CodeLoginReqBody(
phone=15512103215,verifyCode=123456,picCode=null,mobileCountryCode=86)},

参数2: iEncrypData=[objectObject],

参数3: cls=classcom.bw.zsch.client.android.model.common.api.ClientApiResult
```

可以看到`ClientApiRiskReq`类中目前只有`body`以及`head 的action`是有值的.其他都是null

在`callWeb2`中，new的是`ClientApiRiskReq`类的对象，`ClientApiRiskReq`类是继承于`ClientApiReq`类的；

```
21 public class ClientApiRiskReq<T, R> extends ClientApiReq<T> implements Serializable {
```

在调用`EncryptUtils2.encrypt`函数的时候，传进去的也是`ClientApiRiskReq`类的对象，但是`EncryptUtils2.encrypt`函数接收的是一个`ClientApiReq`类的对象，所以这里有一个多态的应用，向上转型，转为了父类的类型

下边看看这个`EncryptUtils2.encrypt`函数

```
public static void encrypt(ClientApiReq clientApiReq, IEncrypData iEncrypData,
Class cls) {

        if (SystemInfo.isLager() && SystemInfo.JSESSIONID_LAGER != null &&
SystemInfo.key_LAGER != null)
        {
            getData(clientApiReq, iEncrypData, false, cls, new
RequestObject(SystemInfo.key_LAGER, SystemInfo.JSESSIONID_LAGER,
SystemInfo.isLager()));
        } else if (!SystemInfo.isLager() && SystemInfo.JSESSIONID != null &&
SystemInfo.key != null) {
            getData(clientApiReq, iEncrypData, false, cls, new
RequestObject(SystemInfo.key, SystemInfo.JSESSIONID, SystemInfo.isLager()));
        } else {
            EncryptBeforeReqManager.moreRequest(clientApiReq, iEncrypData,
false, cls, SystemInfo.isLager(), 2);
        }
    }
```

这里通过一些判断，分为了三个分支，分别走`getData`或`moreRequest`，我们去hook一下，看看走的是`getData`还是`moreRequest`

`hook-moreRequest`

```
没走这个
```

`hook-getData`

```
[Pixel::四川航空 ]-> EncryptUtils2.getData is called:
```

```
clientApiReq=ClientApiRiskReq{head=ReqHead{proVersion='null', path='null',
action='clientapi/login/member/codeLogin', transActionId='null',
timestamp='null', verify='null', sign='null', channelId='null', platformId=null,
imie='null', macAddress ='null', uuid='null', idfa='null', jpushId='null',
appVersion='null', sessionId='null', tokenId='null', acounttype='null',
callWebAction='null', clientIp='null', checkKey='null', customerId='null',
externalChannel='null'}, body=CodeLoginReqBody(phone=15512103215,
verifyCode=123456, picCode=null, mobileCountryCode=86),
riskReqBody=TPSelfChannelRiskControllerSystemReqBody{validType='0',
funcType='null', constId='67c50a93civY8g1rBNefT3ydmd8L3Ie18Eb409d3',
token='null', ip='null', phoneNumber='null', userId='null', userName='null',
source='null', extCurrentUrl='null', extCookie='null', extSessionId='null',
extAccountExist='null', extPasswordCorrect='null', extIdState='null',
extLoginType='null', extId='null', extIdNo='null', email='null',
memberLevel=null, serchFlightRequest=null, submitOrderRequest=null,
activityRequest=null, payRequest=null, refundTicketRequest=null}}
ClientApiReq{head=ReqHead{proVersion='null', path='null',
action='clientapi/login/member/codeLogin', transActionId='null',
timestamp='null', verify='null', sign='null', channelId='null', platformId=null,
imie='null', macAddress='null', uuid='null', idfa='null', jpushId='null',
appVersion='null', sessionId='null', tokenId='null', acounttype='null',
callWebAction='null', clientIp='null', checkKey='null', customerId='null',
externalChannel='null'}, body=CodeLoginReqBody(phone=15512103215,
verifyCode=123456, picCode=null, mobileCountryCode=86)},

iEncrypData=[object Object],

z=false,

cls=class com.bw.zsch.client.android.model.common.api.ClientApiResult,

requestObject=RequestObject
[key=YfxMVBfhW1Q3XTo++q7mdH92jRuWCBtiQ1sVh88IPBN7JmRY28br6UziMIMyDTIz7y92FLDQO52
LNflAVzL/gWdlA2VttidfA84XR4ih+Vhr/RDKuCETK8uy5cyFlOz/QPxWNtTU/GUfKNU8BmPbR6E6Rnq
s5naXATvcZv4xzBI=, sessionid=EBFA757F309486D3D71A7A88496CED5D, islager=false]

//调用栈
java.lang.Throwable
        at
com.bw.zsch.client.android.service.encrypt.EncryptUtils2.getData(Native Method)
        at
com.bw.zsch.client.android.service.encrypt.EncryptUtils2.encrypt(EncryptUtils2.j
ava:41)
        at
com.bw.zsch.client.android.service.encrypt.EncryptUtils2.encrypt(Native Method)
        at
com.bw.zsch.client.android.service.ServiceDataImpl.callWeb2(ServiceDataImpl.java
:4444)
        at
com.bw.zsch.member.login.LoginActivity.lambda$doVerifyCodeLogin$16$com-bw-zsch-
member-login-LoginActivity(LoginActivity.java:493)
        at
com.bw.zsch.member.login.LoginActivity$$ExternalSyntheticLambda0.onToken(Unknown
Source:4)
        at com.bw.zsch.dxrisk.DXRiskUtils$1.run(DXRiskUtils.java:69)
```

确实走了这个`getData`，然后可以看到是从`encrypt`这里调用的.

那我们现在去看下这个`getData`函数

```
public static void getData(final ClientApiReq clientApiReq, final IEncrypData
iEncrypData, final boolean z, final Class<?> cls, RequestObject requestObject) {
        try {
            //以下是为clientApiRiskReq的head去赋值
            clientApiReq.getHead().setChannelId(SystemInfo.getChannelID());
            clientApiReq.getHead().setPlatformId(SystemInfo.getPlatFormId());
            clientApiReq.getHead().setProVersion(SystemInfo.getProVersion());
            clientApiReq.getHead().setImie(SystemInfo.getUniqueDeviceID());
            clientApiReq.getHead().setJpushId(SystemInfo.jpushId);
            clientApiReq.getHead().setCustomerId(SystemInfo.customerId);
            if (SystemInfo.getPlatFormId().intValue() == 0) {

 clientApiReq.getHead().setAppVersion(SystemInfo.getAppVersion());
            } else {

 clientApiReq.getHead().setAppVersion(SystemInfo.getApp_Version());
            }
            clientApiReq.getHead().setMacAddress(SystemInfo.getMacAddress());
            clientApiReq.getHead().setTimestamp(DateTool.getCurrentTime());
            if (requestObject != null && requestObject.getSessionid() != null) {

 clientApiReq.getHead().setSessionId(requestObject.getSessionid());
            }

 clientApiReq.getHead().setTransActionId(UUID.randomUUID().toString());
            if (!requestObject.islager && SystemInfo.TOKENID != null &&
!SystemInfo.TOKENID.equals("")) {
                clientApiReq.getHead().setTokenId(SystemInfo.TOKENID);
            }
            //以上是为clientApiRiskReq的head去赋值



            if (requestObject != null && requestObject.getKey() != null) {

                String encrypt = BwSecurityNative.encrypt(clientApiReq,
requestObject.getKey());
                //加密请求数据，密文为请求体

                final String requestUrl2 =
SystemInfo.getRequestUrl2(clientApiReq.getHead());
                //把参数拼接到请求网址后边，参数为clientApiRiskReq的head的某几个属性



                //这里调用了post方法
                HttpEngine.post(
                requestUrl2, //请求网址
                encrypt,    //加密后的请求体
                new IHttpCallback() {回调逻辑}, //回调
                requestObject //desede的key
```

```
                );
                return;
            }
            iEncrypData.encryptData(null);
        } catch (Exception e) {
            LogUtils.i(e.getMessage());
        }
    }
```

通过刚刚的分析，是在`getData`这个函数中，对url进行了拼接，以及对请求数据进行加密。然后调用`HttpEngine.post`来进行发送请求。

我们可以去hook一下这个`HttpEngine.post`看看它的参数和我们抓包得到的是否一样

```
HttpEngine.post is called:
参数1:
https://mapi.sichuanair.com/zt/tribeport/clientapi/login/member/codeLogin.htm?actionType=clientapi/login/member/codeLogin&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681fc4b10a&networkOperator=&mac=02:00:00:00:00:00&transActionId=59e078be-c363-4723-b8c9-a8771c92af24,

参数2:
lN1TvuqTNNKJPULGeXPd8uZE9JeMRzIaba3/3Z7DiyP/RpwPP62SyUwhhRa9Pm++8WwSbyqgpAgPyRAnQ+Pcfh1NpnzOnMTD5WawvNIM2P4fmZDWsdadcchfYF1ueTiCdULz+k9zNk1+xK5IPL5lmDvga4rhItWqKQcKLf5OFPpmZkQHbKgyqcDeJX+KhLP3aGjr04gdAzECcWkBOHNxR2svPuenvw5wxZfnVJywtKRGASdoO+go0Nz/bPmHhECczFmke04vdRRz21p0TmCwr3sN7hq0EOLj51IGB1oT2V5k6JhcXr4jpTGwjXImeuDrLbO7KUMVCPDx7GFTcG5l/WOMN8mAlGn8KJCEBrWUWFCC5CobA7UXRwpcgi0CvPFkbmfhO2FIdO1Ursir40NnMmLWCNHmuoBheOgu9iC3uv8SZ6TrHf+nwPzqWEIxU/SnffQSRsWFWUM4ws9FmuGW5WHrDuOrEOdc0POQ5iLxjx4NoXt/Ju4o9Q+jSsEzt+2B8LMuogh7tERJLNaH1fBFYB08BMho/CVaniKbGZCEu76LIXZFcp+54vuHePASJ3KKWz9FSV/OFzoXZ1KaD0XvC1F4/U+Tj7K6l9CWUdGPj0SU2nrwn5mbEaF5cMejwZ8JYJL+qwddChCaSdGxEnGe2MCeN+Mu07xIGChurWD/wVi/Xxp2T1XmsHhD244q1MtBW89mBMyWfqttBJOI2AoZvhdqkEOkXqErOfgn04Vs8w71CEwDrk2LGHJsc8p0RWA7jEhTHrhzwgmGuQuKUVYEIVzT3fn4wf+zmOBhC2autcWVJXKyFDb4CXBuqAsq9ExNz7zhHX2MZ8vxUd2qhzhrXiZgKxRmAqsW8lpD0JTSKeBdgMrQ9knwfzVh7DSmQnSXENw2U6UGB2aIGsVnV8U9x/FChNJ3AAl40NTXyo2KXxMsU+4/agA3vbhykBgVpBZhlIGd597CgchCfOFedOqKJAp4Am4gWlP6MrStb4L7KwO8Ix1Iz53cITJfRSZnUTkQzvu4adw1IZ5M2/zblvswVmDOz+jr4axdOOHFXFPDy6gTOLYfEbOG/ItWHxflC8GmPMrxw+bd7Y=
参数3:
iHttpCallback=com.bw.zsch.client.android.service.encrypt.EncryptUtils2$1@33cc81e,requestObject=RequestObject[key=j6VgasM76sYNOp5OBorz3aigO3tvERYpLvC6YX6Du+6LfIIlMA+6UOHZm+dHqg9OPOzwHqzL5tuQydtktJ7djqW71mPcdAo9AT/ekr2mgVYcCUNVsSWU131O0fQI4IKpP/ZwV8+1xBeBYE843zaTAs6VQcYGcEHZqkmmQ1k+Ug0=,sessionid=4F39F3AF84ED0F5C63FD77E79DAEFC30, islager=false]
```

再对比我们刚刚抓包得到的请求：

```
抓包得到的：
lN1TvuqTNNKJPULGeXPd8uZE9JeMRzIaba3/3Z7DiyP/RpwPP62SyUwhhRa9Pm++8WwSbyqgpAgPyRAn
Q+Pcfh1NpnzOnMTD5WawvNIM2P4fmZDWsdadcchfYF1ueTiCdULz+k9zNk1+xK5IPL5lmDvga4rhItWq
KQcKLf5OFPpmZkQHbKgyqcDeJX+KhLP3aGjr04gdAzECcWkBOHNxR2svPuenvw5wxZfnVJywtKRGASdo
O+go0Nz/bPmHhECczFmke04vdRRz21p0TmCwr3sN7hq0EOLj51IGB1oT2V5k6JhcXr4jpTGwjXImeuDr
LbO7KUMVCPDx7GFTcG5l/WOMN8mAlGn8KJCEBrWUWFCC5CobA7UXRwpcgi0CvPFkbmfhO2FIdO1Urrsi
r40NnMmLWCNHmuoBheOgu9iC3uv8SZ6TrHf+nwPzqWEIxU/SnffQSRsWFWUM4ws9FmuGW5WHrDuOrEOd
c0POQ5iLxjx4NoXt/Ju4o9Q+jSsEzt+2B8LMuogh7tERJLNaH1fBFYB08BMho/CVaniKbGZCEu76LIXZ
Fcp+54vuHePASJ3KKWz9FSV/OFzoXZ1KaD0XvC1F4/U+Tj7K6l9CWUdGPj0SU2nrwn5mbEaF5cMejwZ8
JYJL+qwddChCaSdGxEnGe2MCeN+Mu07xIGChurWD/wVi/Xxp2T1XmsHhD244q1MtBW89mBMyWfqttBJO
I2AoZvhdqkEOkXqErOfgn04Vs8w71CEwDrk2LGHJsc8p0RWA7jEhTHrhzwgmGuQuKUVYEIVzT3fn4wf+
zmOBhC2autcWVJXKyFDb4CXBuqAsq9ExNz7zhHX2MZ8vxUd2qhzhrXiZgKxRmAqsW8lpD0JTSKeBdgMr
Q9knwfzVh7DSmQnSXENw2U6UGB2aIGsVnV8U9x/FChNJ3AAl40NTXyo2KXxMsU+4/agA3vbhykBgVpBZ
hlIGd597CgchCfOFed0qKJAp4Am4gWlP6MrStb4L7KwO8Ix1Iz53cITJfRSZnUTkQzvu4adw1IZ5M2/z
blvswVmDOz+jr4axd00HFXFPDy6gTOLYfEbOG/ItWHxflC8GmPMrxw+bd7Y=

hook post方法得到的 post 的第二个参数：
lN1TvuqTNNKJPULGeXPd8uZE9JeMRzIaba3/3Z7DiyP/RpwPP62SyUwhhRa9Pm++8WwSbyqgpAgPyRAn
Q+Pcfh1NpnzOnMTD5WawvNIM2P4fmZDWsdadcchfYF1ueTiCdULz+k9zNk1+xK5IPL5lmDvga4rhItWq
KQcKLf5OFPpmZkQHbKgyqcDeJX+KhLP3aGjr04gdAzECcWkBOHNxR2svPuenvw5wxZfnVJywtKRGASdo
O+go0Nz/bPmHhECczFmke04vdRRz21p0TmCwr3sN7hq0EOLj51IGB1oT2V5k6JhcXr4jpTGwjXImeuDr
LbO7KUMVCPDx7GFTcG5l/WOMN8mAlGn8KJCEBrWUWFCC5CobA7UXRwpcgi0CvPFkbmfhO2FIdO1Urrsi
r40NnMmLWCNHmuoBheOgu9iC3uv8SZ6TrHf+nwPzqWEIxU/SnffQSRsWFWUM4ws9FmuGW5WHrDuOrEOd
c0POQ5iLxjx4NoXt/Ju4o9Q+jSsEzt+2B8LMuogh7tERJLNaH1fBFYB08BMho/CVaniKbGZCEu76LIXZ
Fcp+54vuHePASJ3KKWz9FSV/OFzoXZ1KaD0XvC1F4/U+Tj7K6l9CWUdGPj0SU2nrwn5mbEaF5cMejwZ8
JYJL+qwddChCaSdGxEnGe2MCeN+Mu07xIGChurWD/wi/Xxp2T1XmsHhD244q1MtBW89mBMyWfqttBJOI
2AoZvhdqkEOkXqErOfgn04Vs8w71CEwDrk2LGHJsc8p0RWA7jEhTHrhzwgmGuQuKUVYEIVzT3fn4wf+z
mOBhC2autcWVJXKyFDb4CXBuqAsq9ENz7zhHX2MZ8vxUd2qhzhrXiZgKxRmAqsW8lpD0JTSKeBdgMrQ9
knwfzVh7DSmQnSXENw2U6UGB2aIGsVnV8U9x/FChNJ3AAl40NTXyo2KXxMsU+4/agA3vbhykBgVpBZhl
IGd597CgchCfOFed0qJAp4Am4gWlP6MrStb4L7KwO8Ix1Iz53cITJfRSZnUTkQzvu4adw1IZ5M2/zblv
swVmDOz+jr4axd00HFXFPDy6gTOLYfEbOG/ItWHxflC8GmPMrxw+bd7Y=
```

文本比对工具

[全屏]

```
1 mDOz+jr4axd00HFXFPDy6gTOLYfEbOG/ItWHxflC8GmPMrxw+bd7Y=
```

```
1 mDOz+jr4axd00HFXFPDy6gTOLYfEbOG/ItWHxflC8GmPMrxw+bd7Y=
```

经过对比，是一样的。

那么就可以确定了，软件在发送请求的时候，会先对请求体进行加密。

是通过 `BwSecurityNative.encrypt` 来对请求体进行加密的。

我们去看看 `BwSecurityNative.encrypt`

```
public static native String encrypt(ClientApiReq clientApiReq, String str);
```

下边，我们去把这个加密函数解出来，解出来后就能对请求体进行解密了。并且我们在这还看到一个名为 `decrypt`，很可能是对响应体解密的。

我们现在去找这个native函数看看

关于encrypt

`可以直接跳到 encrypt总结，去看最终的总结。这里分析的过程有些啰嗦！`

IDA View-A | Hex View-1 | Structures | Enums

Name	Address	Ordinal
Java_com_bw30_zsch_security_BwSecurityNative_decrypt	00003086	
Java_com_bw30_zsch_security_BwSecurityNative_encrypt	000024F0	
Java_com_bw30_zsch_security_BwSecurityNative_reqSign	00003348	
Java_com_bw30_zsch_security_BwSecurityNative_sign	00003050	
Java_org_xdq_aes_util_AES4CUtil_base642Byte	00005454	
Java_org_xdq_aes_util_AES4CUtil_decrypt	000055C0	
Java_org_xdq_aes_util_AES4CUtil_encrypt	00005580	
Java_org_xdq_aes_util_AES4CUtil_getAESKey	00005678	
Java_org_xdq_aes_util_AES4CUtil_setAESKey	00005694	
Java_org_xdq_aes_util_AES4CUtil_string2Base64	00005368	

打开看到是静态注册的：

```
int __fastcall Java_com_bw30_zsch_security_BwSecurityNative_encrypt(
        JNIEnv *env,
        jclass jclass,
        jstring clientApiReq,
        jstring jstr_2)
{
  int v4; // r0
  int JbyteArray; // [sp+4h] [bp-3Ch]
  int v7; // [sp+8h] [bp-38h]
  int v8; // [sp+Ch] [bp-34h]
  int v9; // [sp+10h] [bp-30h]
  int v10; // [sp+14h] [bp-2Ch]
  int v11; // [sp+18h] [bp-28h]
  int v12; // [sp+1Ch] [bp-24h]
  int v13; // [sp+20h] [bp-20h]
  int des_key; // [sp+24h] [bp-1Ch]

  des_key = j_des_key_m((int)env, (int)jstr_2);
  v13 = j_req_sign((int)env, (int)clientApiReq);
  v12 = j_md5_encrypt(env, v13);
  v11 = j_hex_encode(env, v12);
  v10 = j_jstringToJbyteArray(env, v11);
  v9 = j_rsa_encrpy_by_public_key(env, v10);
  v8 = j_hex_encode(env, v9);
  j_set_req_sign(env, clientApiReq, v8);
  v7 = j_to_json(env, clientApiReq);
  JbyteArray = j_jstringToJbyteArray(env, v7);
  v4 = j_desecb_encrypt(env, des_key, JbyteArray);
  return j_base64_encode(env, v4);
}
```

这样看还挺友好的，见明知意

关于v14 = j_des_key_m(env, jstr_2);

```
des_key = j_des_key_m((int)env, (int)jstr_2);
```

```
里边有两条逻辑，如下：
    v2 = j_base64_decode(env, jstr_2);
    return j_rsa_decrypt_by_private_key(env, v2);

其中j_base64_decode的逻辑就是通过jni函数调用java层的base64的decode：
        v4 = (*env)->FindClass(env,
"com/bw/zsch/client/android/service/util/Base64");
```

```
        v2 = (*env)->GetStaticMethodID(env, v4, "decode", "(Ljava/lang/String;)
[B");
        return (int)(*env)->CallStaticObjectMethod(env, v4, v2, jstr_2);
```

虽然调用的不是系统的base64，但经过验证，这是一个标准的base64解码函数

接下来是j_rsa_decrypt_by_private_key(env, v2);

把env和v2传进去了，v2是我们对java层传过来的第二个参数进行base64解码后的结果

进去后先获取了private_key,经静态分析及hook验证确认private_key如下：

MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAJwHM1FGXfkNFuiV
TwJcabu02KO2dRMDEHbyJ78OJpSw+vrp6JPXSdXd/NiN6TyxCfclulFYxf+eUj7I
jR89Io9CbNmaynUyWfAkN7I/Ov6lNQRN0g5OnHEh2Nt5SFVuHnSrymH+QbW5ACkb
k//Sr45oL5pyILtLJS+Iaw29N+8pAgMBAAECgYEAhYXT8LWbLcp51EgR+R9EjJHS
4yb9QWvZ5zCLuyDR82URIzCRDBOs6AyOmE25T2rh8FKlA4BIONuX3oqZ2iXjMAe4
2jhIVLj5qEI7xlOPGUnaHpi3mS3Cr4445eqzymt39GHNz/BVqjqR5noMMAQetKc/
Ouc3YFz1LhO1c/eM+FECQQDK9tIozBbXhrLbGryYpmu/GSeLhiwI5yQlRvYGOw1z
HjtY+cpgYkHes/gyv2QvWsSv57UcoOfk9sZdSx8rluJ7AkEAxMyelSEXYPL3fAkS
Ot5O+TpetXNnjUgOHnk6iPjrAzAia2YVXsh0ICKYA7Z9lVnlkrimQXyc4Wr/ELUD
O5LFqwJBAKN5ndhCWfuKgLC9nj4NfSdUaWeV0A3zrkdgGnT2R2T7aq3Fs91iBtAL
gNPPGDCUGWvdYyEH2rZulWM7SIXhZ9UCQCWElN79VgWnjELbP/lvZELjO9b/zOgB
6p8ooV4nhjYvYQB0j7JoKKtPrQdYLQzh/IBCL3/gj8/xo/q9Z4Zd7sOCQAMPp+//
yb/ZkL2ksIIj5MtJHoYDcXZAe6aZvXNnkCxUwPwqdSocU9Y8Xkf+vUH6flOtUWBn
lrdb3xOtHLXOGKM=

对这个j_rsa_decrypt_by_private_key(env, v2);进行了分析，总的来说，就是一个RSA解密的函数，把从java层传过来的第二个参数，解密，其私钥为MIIC...OGKM=
解密后返回

总结v14 = j_des_key_m(env, jstr_2)

```
int __fastcall des_key_m(JNIEnv *env, jstring jstr_2)
{
  int base64_decode_result; // r0

  base64_decode_result = j_base64_decode(env, jstr_2);
  return j_rsa_decrypt_by_private_key(env, base64_decode_result);
}
```

在这个里边主要是由两条逻辑：

一个是base64解码：

```
int __fastcall base64_decode(JNIEnv *env, int jstr_2)
{
  _jmethodID *v2; // r0
  jclass v4; // [sp+Ch] [bp-14h]

  v4 = (*env)->FindClass(env, "com/bw/zsch/client/android/service/util/Base64");
  v2 = (*env)->GetStaticMethodID(env, v4, "decode", "(Ljava/lang/String;)[B");
  return (*env)->CallStaticObjectMethod(env, v4, v2, jstr_2);
}
```

base64就是直接调用的java层的，通过验证是标准的base64解码

一个是rsa解密：

```
1 int __fastcall rsa_decrypt_by_private_key(JNIEnv *env, int base64_decode_result)
2 {
3   size_t private_key_len; // r0
4   _jmethodID *doFinal_id; // r0
5   _jmethodID *Cipher_init; // [sp+2Ch] [bp-B4h]
6   jobject Cipher_Instance; // [sp+30h] [bp-B0h]
7   _jmethodID *Cipher_getinstance_id; // [sp+34h] [bp-ACh]
8   jclass Cipher_class; // [sp+38h] [bp-A8h]
9   jobject PrivateKey; // [sp+3Ch] [bp-A4h]
10  _jmethodID *generatePrivate_id; // [sp+40h] [bp-A0h]
11  jobject KeyFactory_Instance; // [sp+44h] [bp-9Ch]
12  jstring jstr_RSA; // [sp+48h] [bp-98h]
13  _jmethodID *KeyFactory_getInstance_id; // [sp+4Ch] [bp-94h]
14  jclass KeyFactory_class; // [sp+50h] [bp-90h]
15  jobject PKCS8EncodedKeySpec_obj; // [sp+54h] [bp-8Ch]
16  _jmethodID *PKCS8EncodedKeySpec_init_id; // [sp+58h] [bp-88h]
17  jclass PKCS8EncodedKeySpec_class; // [sp+5Ch] [bp-84h]
18  int private_key; // [sp+60h] [bp-80h]
19  jstring private_key_base64; // [sp+64h] [bp-7Ch]
20  char *ptr; // [sp+68h] [bp-78h]
21  int v21; // [sp+6Ch] [bp-74h]
22  size_t v24; // [sp+C8h] [bp-18h]
23  char *v25; // [sp+CCh] [bp-14h]
```

首先这一块是初始化一些变量的

```
24
25  private_key_len = _strlen_chk(::private_key[0], 0xFFFFFFFF);// 获取private_key的长度
26  ptr = malloc(private_key_len + 1);                  // 开辟private_key这么大的空间
27  _strncpy_chk(ptr, ::private_key[0], 10, -1);  // 从private[0]开始复制10位到ptr中
28  ptr[10] = 0;                                        // 为ptr添加\0
29  _strncat_chk(ptr, ::private_key[0] + 20, 10, -1);// 从private[20]的位置开始，往ptr中追加
30  ptr[20] = 0;                                        // 为ptr添加\0
31  _strncat_chk(ptr, ::private_key[0] + 10, 10, -1);
32  ptr[30] = 0;
33  v25 = ::private_key[0] + 30;
34  v24 = _strlen_chk(::private_key[0], 0xFFFFFFFF) - 30;// private_key的长度减去30 30是已经复制进ptr的个数
35  _strncat_chk(ptr, v25, v24, -1);                    // 把剩下的复制进ptr
36                                                      //
37  private_key_base64 = (*env)->NewStringUTF(env, ptr);// 这里就是去取到了private_key,先取前10个，然后取10-20，然后取20-30，最后把剩下的全取过来
38  free(ptr);
39  private_key = j_base64_decode(env, private_key_base64);// 拿到字节形式的私钥
```

这里是从privete_key数组中拿私钥，拿到后进行base64解码

```
40  PKCS8EncodedKeySpec_class = (*env)->FindClass(env, "java/security/spec/PKCS8EncodedKeySpec");// 找PKCS8EncodedKeySpec类
41  PKCS8EncodedKeySpec_init_id = (*env)->GetMethodID(env, PKCS8EncodedKeySpec_class, "<init>", "([B)V");// 获取PKCS8EncodedKeySpec的初始化方法id
42  PKCS8EncodedKeySpec_obj = (*env)->NewObject(env, PKCS8EncodedKeySpec_class, PKCS8EncodedKeySpec_init_id, private_key);// 创建PKCS8EncodedKeySpec对象
43  KeyFactory_class = (*env)->FindClass(env, "java/security/KeyFactory");// 找KeyFactory类
44  KeyFactory_getInstance_id = (*env)->GetStaticMethodID(
45                                 env,
46                                 KeyFactory_class,
47                                 "getInstance",
48                                 "(Ljava/lang/String;)Ljava/security/KeyFactory;");// 调用KeyFactory的getInstance方法id
49  if ( KeyFactory_getInstance_id )
50  {
51    jstr_RSA = (*env)->NewStringUTF(env, "RSA");// 为获取实例做准备
52    KeyFactory_Instance = (*env)->CallStaticObjectMethod(env, KeyFactory_class, KeyFactory_getInstance_id, jstr_RSA);// 调用KeyFactory的getInstance方法
53    generatePrivate_id = (*env)->GetMethodID(
54                           env,
55                           KeyFactory_class,
56                           "generatePrivate",
57                           "(Ljava/security/spec/KeySpec;)Ljava/security/PrivateKey;");// 获取类PrivateKey下generatePrivate的方法id
58    PrivateKey = (*env)->CallObjectMethod(env, KeyFactory_Instance, generatePrivate_id, PKCS8EncodedKeySpec_obj);// 调用类PrivateKey下generatePrivate的
59    Cipher_class = (*env)->FindClass(env, "javax/crypto/Cipher");
60    Cipher_getinstance_id = (*env)->GetStaticMethodID(
61                              env,
62                              Cipher_class,
63                              "getInstance",
64                              "(Ljava/lang/String;)Ljavax/crypto/Cipher;");
65    Cipher_Instance = (*env)->CallStaticObjectMethod(env, Cipher_class, Cipher_getinstance_id, jstr_RSA);
66    Cipher_init = (*env)->GetMethodID(env, Cipher_class, "init", "(ILjava/security/Key;)V");
67    (*env)->CallVoidMethod(env, Cipher_Instance, Cipher_init, 2, PrivateKey);
68    doFinal_id = (*env)->GetMethodID(env, Cipher_class, "doFinal", "([B)[B");
69    return (*env)->CallObjectMethod(env, Cipher_Instance, doFinal_id, base64_decode_result);
70  }
71  return v21;
72 }
```

这里就是获取RSA实例，初始化为解密模式，用的是刚刚拿到的私钥

然后对v14 = j_des_key_m(env, jstr_2)中的jstr_2进行了解密，解密后传给了v14

关于v13 = j_req_sign(env, clientApiReq);

```
des_key = j_des_key_m((int)env, (int)jstr_2);
v13 = j_req_sign((int)env, (int)clientApiReq);
```

这里传进去的是env和java层传过来的请求数据

```
1 int __fastcall req_sign(JNIEnv *env, int clientApiReq)
2 {
3   jstring str_classesdex; // r0
4   jclass ZipEntry_class; // r0
5   jobject (*CallObjectMethod)(JNIEnv *, jobject, jmethodID, ...); // [sp+20h] [bp-E8h]
6   jmethodID (*GetMethodID)(JNIEnv *, jclass, const char *, const char *); // [sp+28h] [bp-E0h]
7   size_t transActionId_byteArray_len; // [sp+40h] [bp-C8h]
8   int v8; // [sp+44h] [bp-C4h]
9   void *ptr; // [sp+48h] [bp-C0h]
10  size_t size; // [sp+4Ch] [bp-BCh]
11  _jmethodID *getCrc_id; // [sp+5Ch] [bp-ACh]
12  jobject call_getEntry_classesdex; // [sp+60h] [bp-A8h]
13  jmethodID getEntry_id; // [sp+64h] [bp-A4h]
14  jobject ZipFile_obj; // [sp+68h] [bp-A0h]
15  _jmethodID *ZipFile_init_id; // [sp+6Ch] [bp-9Ch]
16  jclass ZipFile_class; // [sp+70h] [bp-98h]
17  jobject codePath; // [sp+74h] [bp-94h]
18  _jfieldID *codePath_id; // [sp+78h] [bp-90h]
19  jclass SystemInfo_class; // [sp+7Ch] [bp-8Ch]
20  _BYTE *timestamp_byteArray; // [sp+80h] [bp-88h]
21  const char *transActionId_byteArray; // [sp+84h] [bp-84h]
22  jobject transActionId; // [sp+88h] [bp-80h]
23  _jfieldID *transActionId_id; // [sp+8Ch] [bp-7Ch]
24  jobject timestamp; // [sp+90h] [bp-78h]
25  _jfieldID *timestamp_id; // [sp+94h] [bp-74h]
26  jclass clientApiReq_class_2; // [sp+98h] [bp-70h]
27  jobject v27; // [sp+9Ch] [bp-6Ch]
28  _jfieldID *v28; // [sp+A0h] [bp-68h]
29  jclass clientApiReq_class; // [sp+A4h] [bp-64h]
30  char v32[16]; // [sp+E8h] [bp-20h] BYREF
31  int v33; // [sp+F8h] [bp-10h]
```

初始化变量

```
33  clientApiReq_class = (*env)->GetObjectClass(env, clientApiReq);
34  v28 = ((*env)->GetFieldID)(env, clientApiReq_class);
35  _android_log_print(4, "testL", "head");
36  v27 = (*env)->GetObjectField(env, clientApiReq, v28);
37  clientApiReq_class_2 = (*env)->GetObjectClass(env, v27);
38  timestamp_id = ((*env)->GetFieldID)(env, clientApiReq_class_2, "timestamp");
39  timestamp = (*env)->GetObjectField(env, v27, timestamp_id);
40  transActionId_id = (*env)->GetFieldID(env, clientApiReq_class_2, "transActionId", "Ljava/lang/String;");
41  transActionId = (*env)->GetObjectField(env, v27, transActionId_id);
42  transActionId_byteArray = j_jstringTostring(env, transActionId);
43  timestamp_byteArray = j_jstringTostring(env, timestamp);
44  SystemInfo_class = (*env)->FindClass(env, "com/bw/zsch/client/android/service/util/SystemInfo");
45  codePath_id = (*env)->GetStaticFieldID(env, SystemInfo_class, "codePath", "Ljava/lang/String;");
46  codePath = (*env)->GetStaticObjectField(env, SystemInfo_class, codePath_id);
47  ZipFile_class = (*env)->FindClass(env, "java/util/zip/ZipFile");
48  ZipFile_init_id = (*env)->GetMethodID(env, ZipFile_class, "<init>", "(Ljava/lang/String;)V");
49  ZipFile_obj = (*env)->NewObject(env, ZipFile_class, ZipFile_init_id, codePath);
50  getEntry_id = (*env)->GetMethodID(env, ZipFile_class, "getEntry", "(Ljava/lang/String;)Ljava/util/zip/ZipEntry;");
51  CallObjectMethod = (*env)->CallObjectMethod;
52  str_classesdex = (*env)->NewStringUTF(env, "classes.dex");
53  call_getEntry_classesdex = CallObjectMethod(env, ZipFile_obj, getEntry_id, str_classesdex);
54  GetMethodID = (*env)->GetMethodID;
55  ZipEntry_class = (*env)->GetObjectClass(env, call_getEntry_classesdex);
56  getCrc_id = GetMethodID(env, ZipEntry_class, "getCrc", "()J");
57  (*env)->CallLongMethod(env, call_getEntry_classesdex, getCrc_id);
58  memset(v32, 0, sizeof(v32));                    // 把v32清空
59  v33 = 0;
60  sub_33DC(v32, 20, "%lld", 2822563731LL);
61  _android_log_print(4, "testL", "%s", v32);
62  transActionId_byteArray_len = _strlen_chk(transActionId_byteArray, 0xFFFFFFFF);
63  size = transActionId_byteArray_len + _strlen_chk(v32, 0x14u) + 11;
64  ptr = malloc(size);
```

```
65  if ( !ptr )
66    exit(1);
67  _strlcpy_chk(ptr, timestamp_byteArray, 11, -1);
68  _strcat_chk(ptr, transActionId_byteArray, -1);
69  _strcat_chk(ptr, v32, -1);
70  v8 = j_stoJbyteArray(env, ptr);
71  (*env)->DeleteLocalRef(env, ZipFile_class);
72  (*env)->DeleteLocalRef(env, call_getEntry_classesdex);
73  (*env)->DeleteLocalRef(env, clientApiReq_class);
74  (*env)->DeleteLocalRef(env, clientApiReq_class_2);
75  (*env)->DeleteLocalRef(env, v27);
76  free(ptr);
77  return v8;
78 }
```

最后返回的是v8，v8是由下边三次拼接复制得到的

```
_strlcpy_chk(ptr, timestamp_byteArray, 11, -1);
_strcat_chk(ptr, transActionId_byteArray, -1);
_strcat_chk(ptr, v32, -1);
```

对这个`j_req_sign`进行hook，打印返回值发现是：

```
2025-03-071584b7ee-3039-43b7-b9f0-cbc7925e3f142822563731

    第一部分            第二部分                          第三部分
2025-03-07  1584b7ee-3039-43b7-b9f0-cbc7925e3f14    2822563731
```

对比发现，这个返回值是由三部分组成的，也分别对应这三次的复制拼接

第一部分对应的是：`_strlcpy_chk(ptr, timestamp_byteArray, 11, -1);`

可以看到是一个时间戳，并且是只有年月日的，且长度为11(包含\0)

分析：

```
timestamp_id = ((*env)->GetFieldID)(env, clientApiReq_class_2, "timestamp");
timestamp_jstring = (*env)->GetObjectField(env, v27, timestamp_id);
  这里它是调用的clientApiReq类下的timestamp属性，来拿到这个日期的。
  我们去java层看下这个类clientApiReq，看了下发现没有这个属性，后边经过验证是传进来的
clientApiReq的父类，父类李有这个属性，这里就是去拿timestamp的值，然后再拼接的时候，只取了11
位(包含\0)
```

第二部分对应的是：`_strcat_chk(ptr, transActionId_byteArray, -1);`

验证发现是传进来的对象中`head`的`transActionId`

```
 transActionId_id = (*env)->GetFieldID(env, clientApiReq_class_2,
"transActionId", "Ljava/lang/String;");
 transActionId = (*env)->GetObjectField(env, v27, transActionId_id
 transActionId_byteArray = j_jstringTostring(env, transActionId);
 这里是从java层反射拿到这个类中head的transActionId的值，然后拼接到里边
```

第三部分对应的是：`_strcat_chk(ptr, v32, -1);`

这个是固定的：`2822563731`，并且可以看到，它是通过`sub_33DC(v32, 20, "%lld", 2822563731LL);`得到的

```
int sub_33DC(int a1, int a2, int a3, ...)
{
  _DWORD *v3; // r12
  va_list va; // [sp+2Ch] [bp-4h] BYREF

  va_start(va, a3);
  *v3 = va;
  return _vsprintf_chk(a1, 0, a2, a3);
}

这个是计算dex的CRC值，以验证dex的完整性，6.13.1版本对应的值为：2822563731
```

总结v13 = j_req_sign(env, clientApiReq);

传进去的是java传来的请求数据

返回的是一个字符串的字节数组，其由三部分组成：`时间戳的日期部分 + transActionId + dex的CRC值:2822563731`，如下:

```
2025-03-07 1584b7ee-3039-43b7-b9f0-cbc7925e3f14 2822563731

  第一部分            第二部分                                第三部分
2025-03-07   1584b7ee-3039-43b7-b9f0-cbc7925e3f14     2822563731
```

核心功能

1. **提取请求头信息**：
   从 `clientApiReq` 对象中提取 `timestamp`（时间戳）和 `transActionId`（事务ID）字段。
2. **获取应用代码路径**：
   读取 `SystemInfo` 类的静态字段 `codePath`（APK/DEX文件路径）。
3. **计算DEX文件CRC校验值**：
   通过 `ZipFile` 读取 `classes.dex` 的CRC值（唯一标识DEX文件的校验码）。
4. **拼接签名内容**：
   将 `timestamp`、`transActionId` 和 CRC 值拼接成一个字符串，最终转换为字节数组返回。

关于v12 = j_md5_encrypt(env,v13);

传进去的是env和前边通过`j_req_sign`拿到的拼接的字符串的字节数组

```
int __fastcall md5_encrypt(JNIEnv *env, jbyteArray timestamp_transActionId_guding)
{
  jstring MD5; // r0
  _jmethodID *digest_id; // r0
  jmethodID getInstance_id; // [sp+4h] [bp-2Ch]
  jobject (*CallStaticObjectMethod)(JNIEnv *, jclass, jmethodID, ...); // [sp+8h] [bp-28h]
  jobject MD5_Instance; // [sp+14h] [bp-1Ch]
  jclass MessageDigest_class; // [sp+1Ch] [bp-14h]

  _android_log_print(4, "testL", "md5-encrypt");
  MessageDigest_class = (*env)->FindClass(env, "java/security/MessageDigest");
  getInstance_id = (*env)->GetStaticMethodID(
                     env,
                     MessageDigest_class,
                     "getInstance",
                     "(Ljava/lang/String;)Ljava/security/MessageDigest;");
  CallStaticObjectMethod = (*env)->CallStaticObjectMethod;
  MD5 = (*env)->NewStringUTF(env, "MD5");
  MD5_Instance = CallStaticObjectMethod(env, MessageDigest_class, getInstance_id, MD5);
  digest_id = (*env)->GetMethodID(env, MessageDigest_class, "digest", "([B)[B");
  return (*env)->CallObjectMethod(env, MD5_Instance, digest_id, timestamp_transActionId_guding);
}
```

这个比较简单，就是调用MD5对刚刚得到的那个字符串:`时间戳的日期部分 + transActionId + dex的CRC值:2822563731`进行加密

关于v11 = j_hex_encode(env, v12);

```
v12 = j_md5_encrypt(env, v13);
```

传进去的是env和刚刚MD5的结果

```
jobject __fastcall hex_encode(JNIEnv *env, jbyteArray md5_result)
{
  jobject encode16_result; // [sp+Ch] [bp-1Ch]
  _jmethodID *encode16_id; // [sp+10h] [bp-18h]
  jclass Base64_class; // [sp+14h] [bp-14h]

  _android_log_print(4, "testL", "byte to  hex");
  Base64_class = (*env)->FindClass(env, "com/bw/zsch/client/android/service/util/Base64");
  if ( !Base64_class )
    _android_log_print(4, "testL", "clazz null");
  _android_log_print(4, "testL", "byte to  hex 1");
  encode16_id = (*env)->GetStaticMethodID(env, Base64_class, "encode16", "([B)Ljava/lang/String;");
  _android_log_print(4, "testL", "byte to  hex 2");
  encode16_result = (*env)->CallStaticObjectMethod(env, Base64_class, encode16_id, md5_result);
  _android_log_print(4, "testL", "byte to  hex 3");
  return encode16_result;
}
```

这个应该是进行了hex编码，但调用的不是系统的，而是自己写的，我们现在去看看这个是标准的hex编码吗

```
public static byte[] decode16(String str) {
        String upperCase = str.trim().replace(" ", "").toUpperCase(Locale.US);
        int length = upperCase.length() / 2;
        byte[] bArr = new byte[length];
        for (int i = 0; i < length; i++) {
            int i2 = i * 2;
            int i3 = i2 + 1;
            bArr[i] = (byte) (Integer.decode("0x" + upperCase.substring(i2, i3)
+ upperCase.substring(i3, i3 + 1)).intValue() & 255);
        }
        return bArr;
    }
```

是标准的hex，所以这里就是对MD5的结果进行hex编码的

关于v10 = j_jstringToJbyteArray(env, v11);

```
v10 = j_jstringToJbyteArray(env, v11);
```

这里就是把MD5后的结果的hex编码进行了getBytes

```
int __fastcall jstringToJbyteArray(JNIEnv *env, jstring a2)
{
  _jmethodID *v2; // r0
  jstring v4; // [sp+8h] [bp-18h]
  jclass v5; // [sp+Ch] [bp-14h]

  v5 = (*env)->FindClass(env, "java/lang/String");
  v4 = (*env)->NewStringUTF(env, "utf-8");
  v2 = (*env)->GetMethodID(env, v5, "getBytes", "(Ljava/lang/String;)[B");
  return (*env)->CallObjectMethod(env, a2, v2, v4);
}
```

关于v9 = j_rsa_encrpy_by_public_key(env, v10);

```
v9 = j_rsa_encrpy_by_public_key(env, v10);
```

传进去的是env和MD5加密然后hex编码再转byte数组的值

```
1  int __fastcall rsa_encrpy_by_public_key(JNIEnv *env, jbyteArray md5_hex_jbytearray_result)
2  {
3    size_t public_key_len; // r0
4    _jmethodID *v3; // r0
5    _jmethodID *v5; // [sp+20h] [bp-A8h]
6    jobject v6; // [sp+24h] [bp-A4h]
7    _jmethodID *v7; // [sp+28h] [bp-A0h]
8    jclass v8; // [sp+2Ch] [bp-9Ch]
9    jobject v9; // [sp+30h] [bp-98h]
10   _jmethodID *v10; // [sp+34h] [bp-94h]
11   jobject v11; // [sp+38h] [bp-90h]
12   jstring RSA; // [sp+3Ch] [bp-8Ch]
13   _jmethodID *X509EncodedKeySpec_instance; // [sp+40h] [bp-88h]
14   jclass KeyFactory_class; // [sp+44h] [bp-84h]
15   jobject v15; // [sp+48h] [bp-80h]
16   int public_key_byteayyar; // [sp+4Ch] [bp-7Ch]
17   jstring public_key_; // [sp+50h] [bp-78h]
18   char *ptr; // [sp+54h] [bp-74h]
19   _jmethodID *X509EncodedKeySpec_init_id; // [sp+58h] [bp-70h]
20   jclass X509EncodedKeySpec_class; // [sp+5Ch] [bp-6Ch]
21   int v21; // [sp+60h] [bp-68h]
```

初始化参数

```
22
23   X509EncodedKeySpec_class = (*env)->FindClass(env, "java/security/spec/X509EncodedKeySpec");
24   X509EncodedKeySpec_init_id = (*env)->GetMethodID(env, X509EncodedKeySpec_class, "<init>", "([B)V");
25   public_key_len = _strlen_chk(public_key[0], 0xFFFFFFFF);
26   ptr = malloc(public_key_len + 1);
27   _strncpy_chk(ptr, public_key[0], 10, -1);
28   ptr[10] = 0;
29   _strncat_chk(ptr, public_key[0] + 15, 5, -1);
30   ptr[15] = 0;
31   _strncat_chk(ptr, public_key[0] + 10, 5, -1);
32   ptr[20] = 0;
33   _strcat_chk(ptr, public_key[0] + 20, -1);
34   public_key_ = (*env)->NewStringUTF(env, ptr);
35   free(ptr);
36   public_key_byteayyar = j_base64_decode(env, public_key_);
```

获取公钥

经hook，得到公钥的base64编码为：

MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+QkdMYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB

```
37   v15 = (*env)->NewObject(env, X509EncodedKeySpec_class, X509EncodedKeySpec_init_id, public_key_byteayyar);
38   KeyFactory_class = (*env)->FindClass(env, "java/security/KeyFactory");
39   X509EncodedKeySpec_instance = (*env)->GetStaticMethodID(
40                                   env,
41                                   KeyFactory_class,
42                                   "getInstance",
43                                   "(Ljava/lang/String;)Ljava/security/KeyFactory;");
44   if ( X509EncodedKeySpec_instance )
45   {
46     RSA = (*env)->NewStringUTF(env, "RSA");
47     v11 = (*env)->CallStaticObjectMethod(env, KeyFactory_class, X509EncodedKeySpec_instance, RSA);
48     v10 = (*env)->GetMethodID(
49             env,
50             KeyFactory_class,
51             "generatePublic",
52             "(Ljava/security/spec/KeySpec;)Ljava/security/PublicKey;");
53     v9 = (*env)->CallObjectMethod(env, v11, v10, v15);
54     v8 = (*env)->FindClass(env, "javax/crypto/Cipher");
55     v7 = (*env)->GetStaticMethodID(env, v8, "getInstance", "(Ljava/lang/String;)Ljavax/crypto/Cipher;");
56     v6 = (*env)->CallStaticObjectMethod(env, v8, v7, RSA);
57     v5 = (*env)->GetMethodID(env, v8, "init", "(ILjava/security/Key;)V");
58     (*env)->CallVoidMethod(env, v6, v5, 1, v9);
59     v3 = (*env)->GetMethodID(env, v8, "doFinal", "([B)[B");
60     return (*env)->CallObjectMethod(env, v6, v3, md5_hex_jbytearray_result);
61   }
62   return v21;
63 }
```

使用公钥加密

关于v8 = j_hex_encode(env, v9);

```
v8 = j_hex_encode(env, v9);
```

对刚刚RSA加密后的结果进行hex编码

关于j_set_req_sign(env, jobject, v8);

```
j_set_req_sign(env, clientApiReq, v8);
```

```
int __fastcall set_req_sign(JNIEnv *env, int ClientApiRiskReq, int md5_hex_jbytearray_RSAEncrypt_hex_result)
{
  jfieldID v3; // r0
  jclass v5; // [sp+4h] [bp-24h]
  jobject v6; // [sp+8h] [bp-20h]
  _jfieldID *v7; // [sp+Ch] [bp-1Ch]
  jclass v8; // [sp+10h] [bp-18h]

  v8 = (*env)->GetObjectClass(env, ClientApiRiskReq);
  v7 = (*env)->GetFieldID(env, v8, "head", "Lcom/bw/zsch/client/android/model/message/ReqHead;");
  v6 = (*env)->GetObjectField(env, ClientApiRiskReq, v7);
  v5 = (*env)->GetObjectClass(env, v6);
  v3 = (*env)->GetFieldID(env, v5, "sign", "Ljava/lang/String;");
  return ((*env)->SetObjectField)(env, v6, v3, md5_hex_jbytearray_RSAEncrypt_hex_result, v3);
}
```

这里是进行了一个签名，是把拼接的字符串=>MD5加密=>hex编码=>RSA加密=>hex编码=>然后进行签名，给了clientApiReq的head.sign，也就是java层传过来的要加密的数据

代码功能

1. **目标**：
   将 `md5_hex_jbytearray_RSAEncrypt_hex_result`（可能是一个加密后的签名字符串）设置到 `ClientApiRiskReq` 对象的 `head.sign` 字段中。
2. **操作对象结构**：
   - `ClientApiRiskReq` 是一个 Java 类，包含字段 `head`（类型为 `ReqHead`）。
   - `ReqHead` 类包含字段 `sign`（类型为 `String`）。

关于v7 = j_to_json(env, jobject);

```
v7 = j_to_json(env, clientApiReq);
```

```
int __fastcall to_json(JNIEnv *a1, int a2)
{
  _jmethodID *v2; // r0
  jclass v4; // [sp+4h] [bp-14h]

  v4 = (*a1)->FindClass(a1, "com/bw/zsch/client/android/model/util/JsonUtils");
  v2 = (*a1)->GetStaticMethodID(a1, v4, "toJson", "(Ljava/lang/Object;)Ljava/lang/String;");
  return (*a1)->CallStaticObjectMethod(a1, v4, v2, a2, v2);
}
```

这里不是直接转为json的，是先做了一些操作，我们去java层分析一下：

找到了最终转json的逻辑：

```
public static String toJSONString(Object obj, SerializeConfig serializeConfig,
SerializeFilter[] serializeFilterArr, String str, int i, SerializerFeature...
serializerFeatureArr) {
        SerializeWriter serializeWriter = new SerializeWriter(null, i,
serializerFeatureArr);
        try {
            JSONSerializer jSONSerializer = new JSONSerializer(serializeWriter,
serializeConfig);
            if (str != null && str.length() != 0) {
```

```
                jSONSerializer.setDateFormat(str);
                jSONSerializer.config(SerializerFeature.WriteDateUseDateFormat,
true);
            }
            if (serializeFilterArr != null) {
                for (SerializeFilter serializeFilter : serializeFilterArr) {
                    jSONSerializer.addFilter(serializeFilter);
                }
            }
            jSONSerializer.write(obj);
            return serializeWriter.toString();
        } finally {
            serializeWriter.close();
        }
    }
```

大概就是取了head、body、riskReqBody三部分，然后把值为null和空的剔除

关于v6 = j_jstringToJbyteArray(env, v7);

```
JbyteArray = j_jstringToJbyteArray(env, v7);
```

把json结果进行getBytes

关于v4 = j_desecb_encrypt(env, jstr2_decode, v6);

```
v4 = j_desecb_encrypt(env, des_key, JbyteArray);
```

```
int __fastcall desecb_encrypt(JNIEnv *env, int jstr2_decode, int ClientApiRiskReq_json_bytearay)
{
  jstring v3; // r0
  int v4; // r0
  jstring v5; // r0
  _jmethodID *v6; // r0
  jobject (*NewObject)(JNIEnv *, jclass, jmethodID, ...); // [sp+10h] [bp-80h]
  jmethodID v9; // [sp+1Ch] [bp-74h]
  jobject (*v10)(JNIEnv *, jclass, jmethodID, ...); // [sp+20h] [bp-70h]
  jmethodID v11; // [sp+2Ch] [bp-64h]
  jobject (*CallStaticObjectMethod)(JNIEnv *, jclass, jmethodID, ...); // [sp+30h] [bp-60h]
  _jmethodID *v13; // [sp+3Ch] [bp-54h]
  jobject v14; // [sp+40h] [bp-50h]
  jclass v15; // [sp+48h] [bp-48h]
  int v16; // [sp+4Ch] [bp-44h]
  jmethodID v17; // [sp+50h] [bp-40h]
  jclass v18; // [sp+54h] [bp-3Ch]
  jobject v19; // [sp+58h] [bp-38h]
  _jmethodID *v20; // [sp+5Ch] [bp-34h]
  jobject v21; // [sp+60h] [bp-30h]
  jclass v22; // [sp+68h] [bp-28h]
  jobject v23; // [sp+6Ch] [bp-24h]
  _jmethodID *v24; // [sp+70h] [bp-20h]
  jclass v25; // [sp+74h] [bp-1Ch]

```

初始化变量

```
26  _android_log_print(4, "testL", "desecb_enc");
27  v25 = (*env)->FindClass(env, "javax/crypto/spec/DESedeKeySpec");
28  v24 = (*env)->GetMethodID(env, v25, "<init>", "([B)V");
29  v23 = (*env)->NewObject(env, v25, v24, jstr2_decode);
30  v22 = (*env)->FindClass(env, "javax/crypto/SecretKeyFactory");
31  v11 = (*env)->GetStaticMethodID(env, v22, "getInstance", "(Ljava/lang/String;)Ljavax/crypto/SecretKeyFactory;");
32  CallStaticObjectMethod = (*env)->CallStaticObjectMethod;
33  v3 = (*env)->NewStringUTF(env, "DESede");
34  v21 = CallStaticObjectMethod(env, v22, v11, v3);
35  if ( !v21 || !v23 )
36    return 0;
37  v20 = (*env)->GetMethodID(env, v22, "generateSecret", "(Ljava/security/spec/KeySpec;)Ljavax/crypto/SecretKey;");
38  v19 = (*env)->CallObjectMethod(env, v21, v20, v23);
39  v18 = (*env)->FindClass(env, "javax/crypto/spec/IvParameterSpec");
40  v17 = (*env)->GetMethodID(env, v18, "<init>", "([B)V");
41  NewObject = (*env)->NewObject;
42  v4 = j_stoJbyteArray(env, iv);
43  v16 = NewObject(env, v18, v17, v4);
44  v15 = (*env)->FindClass(env, "javax/crypto/Cipher");
45  v9 = (*env)->GetStaticMethodID(env, v15, "getInstance", "(Ljava/lang/String;)Ljavax/crypto/Cipher;");
46  v10 = (*env)->CallStaticObjectMethod;
47  v5 = (*env)->NewStringUTF(env, "desede/CBC/PKCS5Padding");
48  v14 = v10(env, v15, v9, v5);
49  v13 = (*env)->GetMethodID(env, v15, "init", "(ILjava/security/Key;Ljava/security/spec/AlgorithmParameterSpec;)V");
50  (*env)->CallVoidMethod(env, v14, v13, 1, v19, v16);
51  v6 = (*env)->GetMethodID(env, v15, "doFinal", "([B)[B");
52  return (*env)->CallObjectMethod(env, v14, v6, ClientApiRiskReq_json_bytearay);
53 }
```

进行3des加密

```
加密算法：desede/CBC/PKCS5Padding
key：F33EDEAC9994DB34BC10007D  (Utf8)  //后验证为不是固定的，是请求体中sessionId的前24位，是通过向服务器发送请求拿到的
i v：01234567 (Utf8)
```

关于return j_base64_encode(env, v4);

```
return j_base64_encode(env, v4);
```

将3des加密后的结果进行base64编码

encrypt总结

```
可以直接跳到最后一段看简洁版！！！
可以直接跳到最后一段看简洁版！！！
可以直接跳到最后一段看简洁版！！！

在java层的代代码为：public static native String encrypt(ClientApiReq clientApiReq, String str);
传进去的是请求数据以及一段base64编码的字符串(后经验证为desede的key，是经过RSA加密的)

进入native层后是静态注册的，追进去可以看到有这么几条逻辑
  1、     v14 = j_des_key_m(env, str2);
  2、     v13 = j_req_sign(env, clientApiReq);
  3、     v12 = j_md5_encrypt(env, v13);
  4、     v11 = j_hex_encode(env, v12);
  5、     v10 = j_jstringToJbyteArray(env, v11);
  6、     v9 = j_rsa_encrpy_by_public_key(env, v10);
  7、     v8 = j_hex_encode(env, v9);
  8、     j_set_req_sign(env, clientApiReq, v8);
  9、     v7 = j_to_json(env, clientApiReq);
  10、    v6 = j_jstringToJbyteArray(env, v7);
  11、    v4 = j_desecb_encrypt(env, v14, v6);
  12、    return j_base64_encode(env, v4);
```

在1、的时候，是把java层传进来的str进行了base64解码然后又使用私钥进行了RSA解密(私钥在下边)，解密后返回给了v14，这个是后边用作desede加密的key
在2、的时候，是把java层传进来的clientApiReq取了一个timestamp的日期部分和transActionId然后和dex文件CRC值拼接起来的，进而把拼接的字符串返回给了v13
在3、的时候对刚刚拼接的字符串进行了MD5加密，然后把byte数组返回给了v12
在4、的时候对刚刚的MD5的密文进行了hex编码，编码完返回给了v11
在5、的时候又对hex编码进行了getbytes操作，把最终的byte数组给了v10
在6、的时候对v10这个字节数组进行了RSA加密(公钥在下边)，加密后把字节数组给了v9
在7、的时候把RSA加密后的结果进行了hex编码
在8、的时候，为clientApiReq的head.sign属性进行了赋值，其值为RSA加密结果的hex编码。
2345678这几步，总的说就是：把由timestamp的日期部分，transActionId，以及dex文件CRC值组成的字符串，进行一个MD5，然后进行hex编码，再对hex编码进行getByes，然后再使用公钥进行RSA加密，加密后再进行hex编码，然后把hex编码写到head.sign中
在9、的时候，把clientApiReq转为json
在10、的时候，对这个json进行getBytes
在11、的时候进行desede/CBC/PKCS5Padding加密，key：从java层传进来的str，进行base64解码，RSA解密后，前24位用作desede的key，iv：01234567(Utf8)
在12、的时候对desede加密的结果进行base64编码，然后返回

总的来说在encrypt中干了两件事：
1.为clientApiRiskReq中head的sign进行赋值，其值为：timestamp的日期部分，transActionId，以及dex文件CRC值拼接起来=>进行MD5加密=>进行hex编码=>再对hex编码进行getBytes=>再进行RSA加密=>加密后进行hex编码=>然后给了clientApiRiskReq中head的sign
//其中timestam和transActionId在clientApiRiskReq中head下，dex的CRC值在6.13.1版本中为固定值：2822563731

2.把java层传来的clientApiRiskReq进行加密：先转为json=>再进行getBytes=>进行desede/CBC/PKCS5Padding加密，key为encrypt的第二个参数(解base64，解RSA后使用),iv为：01234567(Utf8)=>加密后进行base64编码=>返回给java层

在1、的时候解密所用到的RSA私钥的base64编码：
MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAJwHM1FGXfkNFuiVTwJcabu02KO2dRMD
EHbyJ78OJpSw+vrp6JPXSdXd/NiN6TyxCfclulFYxf+eUj7IjR89Io9CbNmaynUyWfAkN7I/Ov6lNQRN
0g5OnHEh2Nt5SFVuHnSrymH+QbW5ACkbk//Sr45oL5pyILtLJS+Iaw29N+8pAgMBAAECgYEAhYXT8LWb
Lcp51EgR+R9EjJHS4yb9QWvZ5zCLuyDR82URIzCRDBOs6AyOmE25T2rh8FKlA4BIONuX3oqZ2iXjMAe4
2jhIVLj5qEI7xlOPGUnaHpi3mS3Cr4445eqzymt39GHNz/BVqjqR5noMMAQetKc/Ouc3YFz1LhO1c/eM
+FECQQDK9tIozBbXhrLbGryYpmu/GSeLhiwI5yQlRvYGOw1zHjtY+cpgYkHes/gyv2QvWsSv57UcoOfk
9sZdSx8rluJ7AkEAxMyelSEXYPL3fAkS0t5O+TpetXNnjUgOHnk6iPjrAzAia2YVXshOICKYA7Z9lVnl
krimQXyc4Wr/ELUDO5LFqwJBAKN5ndhCWfuKgLC9nj4NfSdUaWeVOA3zrkdgGnT2R2T7aq3Fs91iBtAL
gNPPGDCUGWvdYyEH2rZulWM7SIXhZ9UCQCWElN79VgWnjELbP/lvZELjO9b/zOgB6p8ooV4nhjYvYQB0
j7JoKKtPrQdYLQzh/IBCL3/gj8/xo/q9Z4Zd7sOCQAMPp+//yb/ZkL2ksIIj5MtJHoYDcXZAe6aZvXNn
kCxUwPwqdSocU9Y8Xkf+vUH6flOtUWBnlrdb3xOtHLXOGKM=

在6、的时候加密所用到的RSA公钥的base64编码：
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+QkdM
YKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg7
BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB

那么到现在，我们知道了，encrypt是进行`desede/CBC/PKCS5Padding`加密，key为encrypt的第二个参数(解base64，解RSA后使用),iv为：01234567(Utf8)，加密后进行base64编码，然后返回给java层。

我们现在去hook一下系统加解密函数：

同时也在抓包，拿响应体搜了一下，发现：

```
============================================================
Cipher.init.overload('int', 'java.security.Key', 'java.security.spec. ') is called!
----------解密----------
 desede/CBC/PKCS5Padding init key  Utf8:  CD8093353CBFA3EEDEE54637
 desede/CBC/PKCS5Padding init key  Base64:  Q0Q4MDkzMzUzQ0JGQTNFRURFRTU0NjM3
 desede/CBC/PKCS5Padding init key  Hex:  434438303933333533434246413345454445453534363337
------------------------------------------------------------
 desede/CBC/PKCS5Padding init IV  Utf8:  01234567
 desede/CBC/PKCS5Padding init IV  Base64:  MDEyMzQ1Njc=
 desede/CBC/PKCS5Padding init IV  Hex:  3031323334353637
============================================================
Cipher.doFinal.overload('[B') is called!
 desede/CBC/PKCS5Padding doFinal data  Utf8:  ◆kj◆<◆e◆I`↗'+
 desede/CBC/PKCS5Padding doFinal data  Base64:  r2tq9jyVZQ7jSWAZ3o0nKwAcbzVUD5QphsCTRfkJPhg1i9EpHVUpJqQ3srWHVpRiyjDyDNNzzc1Nn9e0ydPVh3VSz/AlYs5oYpFhC
QA2L6ncCHDrFe7sWGZItDOpzwcVLqEb7UKezuD3Eh4WGqz0ju9HdJWc99LTrEaCUjBFxTycoBYwz9nW9/qVPwshWg0s+m0As5b+hxt3hyBkNfa1SNWFP8WPraJM7nL3Lqk1kcha+e83E1YUh4A8ZU
```

拿请求体搜了一下：

```
============================================================
Cipher.init.overload('int', 'java.security.Key', 'java.security.spec. ') is called!
----------加密----------
 desede/CBC/PKCS5Padding init key  Utf8:  CD8093353CBFA3EEDEE54637
 desede/CBC/PKCS5Padding init key  Base64:  Q0Q4MDkzMzUzQ0JGQTNFRURFRTU0NjM3
 desede/CBC/PKCS5Padding init key  Hex:  434438303933333533434246413345454445453534363337
------------------------------------------------------------
 desede/CBC/PKCS5Padding init IV  Utf8:  01234567
 desede/CBC/PKCS5Padding init IV  Base64:  MDEyMzQ1Njc=
 desede/CBC/PKCS5Padding init IV  Hex:  3031323334353637
============================================================
Cipher.doFinal.overload('[B') is called!
 desede/CBC/PKCS5Padding doFinal data  Utf8:  {"body":{"mobileCountryCode":"86","phone":"15512103215","verifyCode":"123456"},"head":{"action":"client
api/login/member/codeLogin","appVersion":"6.13.1","channelId":"myself","customerId":"","imie":"00000000-6fd2-5a75-ffff-ffffca01fdf4","jpushId":"120c8
3f7611feca6238","macAddress":"02:00:00:00:00:00","platformId":0,"proVersion":"2.0","sessionId":"CD8093353CBFA3EEDEE546376ACECB02","sign":"3237b468c61
e4254ad230d11702b794e77fc6e1fc4468b6050804f9cda8681e39806b3ebd83ef6ae05a0fe6a6f11f57db8687f116e43088411fcddcac9fbfef51f4db06581422fec0e173ef172fb65d1
```

经过多次对比，发现了desede的key是怎么来的了，请看vcr：

```
Terminal  Local  Local (2)  Local (3)  Local (4)  Local (5)  Local (6)  Local (7)  Local (8)  Local (9)  Local (10)
============================================================
Cipher.init.overload('int', 'java.security.Key', 'java.security.spec. ') is called!
----------加密----------
 desede/CBC/PKCS5Padding init key  Utf8:  18F90EA09B4BDF4CEB2F7E4F
 desede/CBC/PKCS5Padding init key  Base64:  MThG0TBFQTA5QjRCREY0Q0VCMkY3RTRG
 desede/CBC/PKCS5Padding init key  Hex:  3138463930454130394234424446344345423246374534 46
------------------------------------------------------------
 desede/CBC/PKCS5Padding init IV  Utf8:  01234567
 desede/CBC/PKCS5Padding init IV  Base64:  MDEyMzQ1Njc=
 desede/CBC/PKCS5Padding init IV  Hex:  3031323334353637
============================================================
Cipher.doFinal.overload('[B') is called!
 desede/CBC/PKCS5Padding doFinal data  Utf8:  {"body":{"mobileCountryCode":"86","phone":"19944015856","verifyCode":"123555"},"head":{"action":"client
api/login/member/codeLogin","appVersion":"6.13.1","channelId":"myself","customerId":"","imie":"00000000-6fd2-5a75-ffff-ffffca01fdf4","jpushId":"120c8
3f7611feca6238","macAddress":"02:00:00:00:00:00","platformId":0,"proVersion":"2.0","sessionId":"18F90EA09B4BDF4CEB2F7E4F83CCD4E6","sign":"8fe199077fb
88fa9eff43d175cff30fc70705e6f1d802a866e0e34e63be255b094b22770819556d86fb329dd5b17a4550599a584cba846a14a96c9afa346bf943da345e1253852635 0636e28a8d3d1ed
```

它是截取了sessionId的前24位，这也对应上了desede的168位密钥

sessionId是变化的，每重启一次软件就会重新得到一个新的

然后在调用encrypt方法前，会先对desede的key进行RSA加密，加密后进行base64编码，然后在调用encrypt时，传过来做第二个参数，到so层的时候再进行base64解码，私钥解密，然后在desede初始化密钥的时候使用

所以，要想对请求体进行解密，我们需要先去抓一下这个sessionId然后去取前24位

登录接口的请求数据分析

在之前的分析过程中，我们说过发送登录请求是在`HttpEngine.post`方法里进行的

所以对`post`方法进行了hook，打印了堆栈：

```
HttpEngine.post is called:


参数1
https://mapi.sichuanair.com/zt/tribeport/clientapi/login/member/codeLogin.htm?
actionType=clientapi/login/member/codeLogin&platformID=0&appVersion=6.13.1&chann
elID=myself&imei=5a9992681fc4b10a&networkOperator=&mac=02:00:00:00:00:00&transAc
tionId=c77d5076-3f00-484d-9a8e-e088f684db82,

参数2
HLYW8GLyRI8JFuVtD6wrIUrKdMUnM7w132BdBaJSV0qcRMR92gj7Yso6KnHDfsfvyYvPKBVNIQ+yJ9Pf
wqY6/56TNgQpMENZaoofN/lE6zykMZQN+MbDriZWq+8WfexalNjf9RrCTy1zGA+E+sK2ytsq/IWNd8SO
9qROP2FmCWBoSKuuHiCEPf37gELOpQoyK4eaWRYTkVPAOil3sJf7OqyMkNDl4aj5g5YjJPNNVYLAfDmo
HTYgfXC9A1RboWlQ4zqn+NcqBg4akxFWwpJ+LbE7outXuLoYwot1PjQl/KYtqHz7drUNTOTkX8UHfVUp
TzluBOr/XYQ2++V8vCvDUulZbzj/HW9IO7B/3ZExlMupK8ZXsBbquI/VKQbsq0YwMwIUUlZm3eDbDprx
uc6IRzYOAi1fw6qN6/5GxwfbWf56l6EThv+Hn8BWwF6v3PstMlVoZHqV6BHV1kR3nkMuJMRLgHPX0Ljg
uA0jxKYFF5zn4X3me76G1mlSjEgnXwKtowiJFyuRidItKB1Rc/GRXk8zZqpj43kyf3/Q+9J3nL2zTH+J
bS2G/+ps0Mpt6DBpDzc29sc4OvrhrUz2tHiFkC4UBpCfMOWEK5NMAU2rFL93lHL7VNWgKutl5n3hoyjT
4rfzTVlRoRXWbRJDhTVUdj8qD1ctnGKISVxg0xRpoHOr2HDxtoy+2jLCp3lciOKKrODBBJrirsiJLuf7
/lXgdFboXpY/wmMCJ3FEtFIiuJeHq0a5X0pdOPWh3DHIXM89uBrk7D8GzWroGfhqAYUm/QmchU+h4R5o
sA1JNT3cuuw8awwDVgRPHmBZ6W3GHkNfR8BWfNwvUD3um11ZpiDMcVbGbGheiO06kXTN7BGa+8oI7eXI
IWcgq0Oek+R58OzsTpofSeS3evjfNPmzun8gUeD+cK7CEDRZuv2+pPdXpYGwgZ+Pfltw4Xpk4ef34Cu3
OZFtHo0YkCQZMets3waSqvw459XemuZMrVE0iuOCr461xOPGciP3mdvBjXv58T/OGuFNinPJigkuj+1G
hgcP+fmbhESOaJAr+uFQOb45H1DTiYCIrqKWNb0q5+ReTvnpE3d0ZRjD3V4=,

参数3
com.bw.zsch.client.android.service.encrypt.EncryptUtils2$1@bcf312,

参数4
RequestObject[key=BolFcBT+odVACEpErQ1hps7gsjmqdSzwjxVHzJF9TOCFi1LGPMQoS9cSXqPytf
T2Ar76CeFdL9v4bRU6y50AKOdlnaj6IH9n8+WSEqFo6CQoHvugilAvYN/3VNjBZYogVYrUUqJ7V5YOWE
nMeFxSLKtA2q9FtvkOYT2bt8s8k50=, sessionid=17963D3EAD17AC4407B1DFFBEFF541FA,
islager=false]

调用堆栈
java.lang.Throwable
    at com.bw.zsch.client.android.service.http.HttpEngine.post(Native Method)
    at
com.bw.zsch.client.android.service.encrypt.EncryptUtils2.getData(EncryptUtils2.j
ava:91)
    at
com.bw.zsch.client.android.service.encrypt.EncryptUtils2.encrypt(EncryptUtils2.j
ava:41)
    at
com.bw.zsch.client.android.service.ServiceDataImpl.callWeb2(ServiceDataImpl.java
:4444)
    at com.bw.zsch.member.login.LoginActivity.lambda$doVerifyCodeLogin$15$com-bw-
zsch-member-login-LoginActivity(LoginActivity.java:501)
```

```
    at
com.bw.zsch.member.login.LoginActivity$$ExternalSyntheticLambda3.data(Unknown
Source:8)
    at
com.bw.zsch.client.android.service.ServiceDataImpl$183.encryptData(ServiceDataIm
pl.java:4447)
    at
com.bw.zsch.client.android.service.encrypt.EncryptUtils2$1.onSuccess(EncryptUtil
s2.java:130)
    at
com.bw.zsch.client.android.service.http.HttpEngine$1.onSuccess(HttpEngine.java:1
06)
    at
com.lidroid.xutils.http.HttpHandler.onProgressUpdate(HttpHandler.java:225)
    at
com.lidroid.xutils.task.PriorityAsyncTask$InternalHandler.handleMessage(Priority
AsyncTask.java:206)
    at android.os.Handler.dispatchMessage(Handler.java:107)
    at android.os.Looper.loop(Looper.java:214)
    at android.app.ActivityThread.main(ActivityThread.java:7356)
    at java.lang.reflect.Method.invoke(Native Method)
    at
com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:492
)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:930)
```

这里的参数1就是我们的请求网址及请求参数，参数2就是我们的请求体，所以我们继续往上追，然后去看我们的请求体和请求网址生成的过程

根据堆栈，继续看`getData`

```
public static void getData(final ClientApiReq clientApiReq, final IEncrypData
iEncrypData, final boolean z, final Class<?> cls, RequestObject requestObject) {
        try {
            //以下是为clientApiRiskReq的head去赋值
            clientApiReq.getHead().setChannelId(SystemInfo.getChannelID());
            clientApiReq.getHead().setPlatformId(SystemInfo.getPlatFormId());
            clientApiReq.getHead().setProVersion(SystemInfo.getProVersion());
            clientApiReq.getHead().setImie(SystemInfo.getUniqueDeviceID());
            clientApiReq.getHead().setJpushId(SystemInfo.jpushId);
            clientApiReq.getHead().setCustomerId(SystemInfo.customerId);
            if (SystemInfo.getPlatFormId().intValue() == 0) {

 clientApiReq.getHead().setAppVersion(SystemInfo.getAppVersion());
            } else {

 clientApiReq.getHead().setAppVersion(SystemInfo.getApp_Version());
            }
            clientApiReq.getHead().setMacAddress(SystemInfo.getMacAddress());
            clientApiReq.getHead().setTimestamp(DateTool.getCurrentTime());
            if (requestObject != null && requestObject.getSessionid() != null) {

 clientApiReq.getHead().setSessionId(requestObject.getSessionid());
```

```
            }

 clientApiReq.getHead().setTransActionId(UUID.randomUUID().toString());
            if (!requestObject.islager && SystemInfo.TOKENID != null &&
!SystemInfo.TOKENID.equals("")) {
                clientApiReq.getHead().setTokenId(SystemInfo.TOKENID);
            }
            //以上是为clientApiRiskReq的head去赋值



            if (requestObject != null && requestObject.getKey() != null) {
                String encrypt = BwSecurityNative.encrypt(clientApiReq,
requestObject.getKey());
                //加密请求数据，密文为请求体
                final String requestUrl2 =
SystemInfo.getRequestUrl2(clientApiReq.getHead());
                //把参数拼接到请求网址后边，参数为clientApiReq的head的某几个属性



                //这里调用了post方法
                HttpEngine.post(
                requestUrl2,
                encrypt,
                new IHttpCallback() {回调逻辑},
                requestObject
                );
                return;
            }
            iEncrypData.encryptData(null);
        } catch (Exception e) {
            LogUtils.i(e.getMessage());
        }
    }
```

可以看到这里在调用post的时候传的参数是`requestUrl2、encrypt`

其中`requestUrl2`是请求网址，`encrypt`是加密后的请求体

我们先去看看`clientApiReq`的结构，这是`getData`的第一个参数，后续都是对`clientApiReq`来加密，拼接，发请求的！

但是之前说了，这里传的是他的子类：`clientApiRiskReq`,所以这里有一个多态的用法

```
public class ClientApiReq<T> implements Serializable {
        protected Map<String, Object> attachements;
        protected T body;
        protected ReqHead head;

        public ReqHead getHead() {
                return this.head;
            }

         //一些set get方法
```

```
            public String toString() {
                return "ClientApiReq{head=" + this.head + ", body=" + this.body
+ TokenCollector.END_TERM;
            }
}



public class ClientApiRiskReq<T, R> extends ClientApiReq<T> implements
Serializable {
    private static final long serialVersionUID = 1443572875916929462L;
    private R riskReqBody;
   //一些set get方法
    @Override
    public String toString() {
        return "ClientApiRiskReq{head=" + this.head + ", body=" + this.body + ",
riskReqBody=" + this.riskReqBody + "} " + super.toString();
    }
}
```

请求网址及参数

我们先看`requestUrl2`：

```
final String requestUrl2 = SystemInfo.getRequestUrl2(clientApiReq.getHead());
```

它传进去的是`clientApiRiskReq下边的head`，然后去`SystemInfo.getRequestUrl2`进行拼接的

`SystemInfo.getRequestUrl2`：

```
public static String getRequestUrl2(ReqHead reqHead) {
        StringBuilder sb = new StringBuilder();
        sb.append(SERVER_URL);
        //SERVER_URL = "https://mapi.sichuanair.com/zt/tribeport/";
        sb.append(TextUtils.isEmpty(reqHead.getPath()) ? "" :
reqHead.getPath());
        //这里是获取head里的Path的值，若为空就给 ""
        sb.append(reqHead.getAction());
        //添加head里的action
        sb.append(".htm?actionType=");
        //添加字符串
        sb.append(TextUtils.isEmpty(reqHead.getCallWebAction()) ?
reqHead.getAction() : reqHead.getCallWebAction());
        sb.append(getINFO());
        sb.append("&transActionId=");
        sb.append(reqHead.getTransActionId());
        return sb.toString();
    }
```

这里我们可以去hook一下`getData`，在进来的时候打印一下head，执行完了再打印一下head。看看`head`的所有信息是不是都在`getData`中添加的、

```
Enter getData heade is => ReqHead{proVersion='null', path='null',
action='clientapi/login/member/codeLogin', transActionId='null',
timestamp='null', verify='null', sign='null', channelId='null', platformId=null,
imie='null', macAddress='null', uuid='null', idfa='null', jpushId='null',
appVersion='null', sessionId='null', tokenId='null', acounttype='null',
callWebAction='null', clientIp='null', checkKey='null', customerId='null',
externalChannel='null'}


leave getData heade is => ReqHead{proVersion='2.0', path='null',
action='clientapi/login/member/codeLogin', transActionId='9bb01874-04bb-42dd-
af15-27867829843c', timestamp='2025-03-08 16:03:35', verify='null',
sign='54a3f392cec106b7155789306fc7a8568c054114de177ac710aac1cb79500b03f83b10a641
4f27113b1057d3b7732df9f7dabe1fe64ee58e2ccfe8f9114b1f346b7f6b0682f88defa1572682d1
c7237eaca8abf06fd6e4af5460406ec30964ae9335c2b1320fe7ed2b82d761118e520ee8ea27a364
3130d1de4d3b4be17e17f5', channelId='myself', platformId=0, imie='00000000-6fd2-
5a75-ffff-ffffca01fdf4', macAddress='02:00:00:00:00:00', uuid='null',
idfa='null', jpushId='120c83f7611feca6238', appVersion='6.13.1',
sessionId='AC74ABA9C8B12473B4711BE23A579222', tokenId='null', acounttype='null',
callWebAction='null', clientIp='null', checkKey='null', customerId='',
externalChannel='null'}
```

在进入的时候只有action是有值的，其他的都是null

在getData函数中设置了：`proVersion`、`transActionId`、`timestamp`、`sign`、`channelId`、`platformId`、`imie`、`macAddress`、`jpushId`、`appVersion`、`sessionId`、`customerId`

我们逐一去看看

proVersion

```
clientApiReq.getHead().setProVersion(SystemInfo.getProVersion());

 public static String getProVersion() {
        return PRO_VERSION;
    }

 public static String PRO_VERSION = "2.0";
```

这里初始值是2.0，在登录场景下还是2.0，它对应的也有set方法

transActionId

```
clientApiReq.getHead().setTransActionId(UUID.randomUUID().toString());


  public void setTransActionId(String str) {
        this.transActionId = str;
    }
```

这个就是个UUID

timestamp

```
 clientApiReq.getHead().setTimestamp(DateTool.getCurrentTime());

 public void setTimestamp(String str) {
        this.timestamp = str;
    }

  public static String getCurrentTime() {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat();
        Date date = new Date();
        simpleDateFormat.applyPattern("yyyy-MM-dd HH:mm:ss");
        return simpleDateFormat.format(date);
    }
```

这个就是获取当前时间，用这种格式：`yyyy-MM-dd HH:mm:ss`

sign

这个咱们在so层分析的时候说过!

也就是那个encrypt方法，encrypt就做了两件事，一件事是签名，另一件事就是加密请求体

所以我们拿之前分析的这个签名

```
把拼接的字符串=>MD5加密=>hex编码=>getBytes=>RSA加密=>hex编码=>然后进行签名

这个字符串是以下三部分组成的：
   第一部分                          第二部分                                第三部分
2025-03-07                1584b7ee-3039-43b7-b9f0-cbc7925e3f14     2822563731
timestamp的日期部分              transActionId                          CRC校验值
```

这个sign虽然在head里，但是在拼接URL时并没有用到，在后边加密请求体的时候是对head和body一起加密的。

我们去复现一下sign

```
# 拼接的明文字符串：2025-03-094a2f8eb2-c59e-486c-8fe5-119008c5d0802822563731
# RSA公钥：
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+QkdM
YKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg7
BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB
```

下边是用java写了生成sign的逻辑

```
import base64
import hashlib
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
import time
import uuid
from Crypto.Cipher import DES3
from Crypto.Util.Padding import pad
import requests
from Crypto.Util.Padding import unpad
import json


def bytes_to_hex(bytes_data):
    return bytes_data.hex()
def rsa_encrypt(plain_text, public_key_str):
    # 解码Base64公钥
    public_key_bytes = base64.b64decode(public_key_str)
    # 加载DER格式的公钥
    public_key = serialization.load_der_public_key(
        public_key_bytes,
        backend=default_backend()
    )

    # 获取RSA公钥参数
    n = public_key.public_numbers().n
    e = public_key.public_numbers().e

    # 计算模长的字节数
    modulus_bytes = (n.bit_length() + 7) // 8

    # 将明文转换为字节并进行填充
    plain_bytes = plain_text.encode('utf-8')

    # 检查长度是否超过模长
    if len(plain_bytes) > modulus_bytes:
        raise ValueError(f"明文过长 ({len(plain_bytes)} > {modulus_bytes} bytes)")

    # 使用零字节在前方填充至模长
    padded = b'\x00' * (modulus_bytes - len(plain_bytes)) + plain_bytes

    # 将填充后的字节转换为整数
    m_int = int.from_bytes(padded, byteorder='big')

    # 执行RSA加密: c = m^e mod n
    c_int = pow(m_int, e, n)

    # 将加密结果转换回字节
    encrypted_bytes = c_int.to_bytes(modulus_bytes, byteorder='big')

    return encrypted_bytes
def getSign(inputStr):
    md5 = hashlib.md5()
```

```
    md5.update(inputStr.encode('utf-8'))
    md5_hex_str = md5.hexdigest()
    public_key_str =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+Qkd
MYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg
7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB"
    encrypted_bytes = rsa_encrypt(md5_hex_str, public_key_str)
    sign_hex = bytes_to_hex(encrypted_bytes)
    return  sign_hex
```

channelId

```
clientApiReq.getHead().setChannelId(SystemInfo.getChannelID());
```

是通过 `SystemInfo.getChannelID()` 去取到的，我们去看下这个方法

```
    public static String getChannelID() {
        Object obj;
        try {
            ApplicationInfo applicationInfo =
CTX.getPackageManager().getApplicationInfo(CTX.getPackageName(), 128);
            return (applicationInfo.metaData == null || (obj =
applicationInfo.metaData.get("ZSCH")) == null) ? "" : obj.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }
    }
```

```
在 Android 开发中，**ChannelID**（渠道 ID）通常有两种不同的含义，具体取决于上下文场景：

---

### **1. 应用分发渠道（App Distribution Channel）**
这是最常见的场景，主要用于 **统计不同渠道来源的用户**（如应用市场、广告平台、官网等）。

#### **用途**：
- 区分用户安装来源，统计不同渠道的下载量、激活量、用户行为等。
- 针对不同渠道做定制化功能（例如不同渠道展示不同的活动页面）。

#### **实现方式**：
在 `AndroidManifest.xml` 中通过 `<meta-data>` 标记渠道标识：
```xml
<application>
    <meta-data
        android:name="CHANNEL"  <!-- 例如 ZSCH、UMENG_CHANNEL -->
        android:value="google_play" />
</application>
```

代码中获取：
```java
public static String getChannel() {
```
```

```
    try {
        ApplicationInfo appInfo = context.getPackageManager()
            .getApplicationInfo(context.getPackageName(),
PackageManager.GET_META_DATA);
        return appInfo.metaData.getString("CHANNEL");
    } catch (Exception e) {
        return "unknown";
    }
}
```

多渠道打包：
通过 Gradle 自动化生成不同渠道的 APK：
```groovy
android {
    flavorDimensions "channel"
    productFlavors {
        google_play {
            dimension "channel"
            manifestPlaceholders = [CHANNEL: "google_play"]
        }
        huawei {
            dimension "channel"
            manifestPlaceholders = [CHANNEL: "huawei"]
        }
    }
}
```

---

2. 通知渠道（Notification Channel）
从 Android 8.0（API 26）开始，系统要求为通知分配 **通知渠道（Notification Channel）**，用户可以通过渠道管理通知权限。

用途：
- 将通知按类型分组（如“营销消息”、“系统通知”）。
- 允许用户按渠道关闭或调整通知优先级。

创建通知渠道：
```java
public void createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        String channelId = "order_updates"; // 渠道唯一标识
        CharSequence channelName = "订单通知"; // 用户看到的渠道名称
        int importance = NotificationManager.IMPORTANCE_HIGH;

        NotificationChannel channel = new NotificationChannel(channelId,
channelName, importance);
        channel.setDescription("订单状态变更通知");

        NotificationManager manager =
getSystemService(NotificationManager.class);
        manager.createNotificationChannel(channel);
    }
```

```
}
```

发送通知：
```java
NotificationCompat.Builder builder = new NotificationCompat.Builder(context,
"order_updates");
builder.setContentTitle("订单提醒")
       .setContentText("您的订单已发货");
```

两种 ChannelID 的区别
| **类型** | **应用分发渠道** | **通知渠道**
    |
|-------------------|-------------------------------|---------------------------
------|
| **用途** | 统计用户来源 | 管理通知分类和权限
  |
| **配置位置** | `AndroidManifest.xml` 或 Gradle | 代码动态创建（Android 8.0+）
    |
| **数据形式** | 字符串（如 `"google_play"`） | 唯一字符串（如
`"order_updates"`）|
| **系统要求** | 无 | Android 8.0+
    |

实际场景示例
场景 1：统计应用市场下载量
```java
// 获取分发渠道 ID
String channel = getChannel();
Analytics.trackInstall(channel); // 上报统计平台
```

场景 2：发送分类通知
```java
// 发送订单通知（需提前创建渠道）
NotificationManager.notify(1, builder.build());
```

注意事项
1. **分发渠道**：
   - 确保渠道 ID 唯一且有意义（如 `huawei`、`xiaomi`）。
   - 使用第三方统计工具（如友盟、Firebase）时，需按文档配置渠道 ID。

2. **通知渠道**：
   - 渠道一旦创建，无法通过代码修改（用户可手动调整）。
   - 合理规划渠道分类，避免过多冗余渠道。

```
这个应该是统计下载渠道的，不同的的应用市场对应不同的字符串
这里我使用的adb install
因从这里返回myself可能是因为是我自己安装的原因
```

platformId

```
 clientApiReq.getHead().setPlatformId(SystemInfo.getPlatFormId());

public static Integer getPlatFormId() {
        return PLATFORMID;
    }

public static Integer PLATFORMID = 0;
```

这里初始值是0，在登录的场景下，返回的也是0

imie

```
clientApiReq.getHead().setImie(SystemInfo.getUniqueDeviceID());

 public static String getUniqueDeviceID() {
        return new UUID(("LT" + Build.BOARD + Build.BRAND + Build.DEVICE +
Build.MANUFACTURER + Build.PRODUCT).hashCode(), -905839116).toString();
    }
```

```
在 Java 中，通过 new UUID(long mostSigBits, long leastSigBits) 构造函数传入固定的参数时，生成的 UUID 是固定的。UUID 的值完全由你传入的两个 long 型参数（高64位 mostSigBits 和低64位 leastSigBits）决定。
```

这里是直接拿的uuid，但是在new uuid的时候给的参数是关于手机的信息取的哈希。

macAddress

```
  clientApiReq.getHead().setMacAddress(SystemInfo.getMacAddress());

 public void setMacAddress(String str) {
        this.macAddress = str;
    }

 public static String getMacAddress() {
        String string = SharedPreferencesUtil.getString(CTX,
"system_mac_address", "");
        if (TextUtils.isEmpty(string)) {
            try {
                string = ((WifiManager)
CTX.getSystemService(TencentLocationListener.WIFI)).getConnectionInfo().getMacAd
dress();
                SharedPreferencesUtil.putString(CTX, "system_mac_address",
string);
                return string;
            } catch (Exception e) {
```

```
                e.printStackTrace();
                return string;
            }
        }
        return string;
    }
```

这里是获取mac地址，这个随机给应该也是可以的

jpushId

```
clientApiReq.getHead().setJpushId(SystemInfo.jpushId);

public static String jpushId = "";
```

看了下这个类里边没有set方法，那么肯定是类名.属性的方式赋值的，所以直接去搜了SystemInfo.jpushId

发现只有一个地方是赋值的，其他地方都是调用的

```
SystemInfo.jpushId = JPushInterface.getRegistrationID(context);

  public static String getRegistrationID(Context context) {
        checkContext(context);
        JCoreHelper.runActionWithService(context, JPushConstants.SDK_TYPE,
"get_rid", null);
        return JCoreHelper.getRegistrationID(context);
    }


 public static String getRegistrationID(Context context) {
        Object onEvent = JCoreManager.onEvent(context, JPushConstants.SDK_TYPE,
4, null, null, new Object[0]);
        return onEvent instanceof String ? (String) onEvent : "";
    }
```

这个也没看太懂是干什么的，我hook 修改了这个字段的值，也不影响发送请求

appVersion

```
if (SystemInfo.getPlatFormId().intValue() == 0) {

 clientApiReq.getHead().setAppVersion(SystemInfo.getAppVersion());
            } else {

 clientApiReq.getHead().setAppVersion(SystemInfo.getApp_Version());
            }


    public void setAppVersion(String str) {
        this.appVersion = str;
    }

  public static String getApp_Version() {
        return APP_VERSION;
    }

public static String APP_VERSION = "2.0";
```

这个if好像没用，无论是否等于0，都是要走这个clientApiReq.getHead().setAppVersion(SystemInfo.getAppVersion());逻辑

这里APP_VERSION默认的是2.0

又仔细看了下，发现一个是getAppVersion()另一个是getApp_Version()。还是有些不同的，这里应该是走了getAppVersion()

```
   public static String getAppVersion() {
        try {
            return CTX.getPackageManager().getPackageInfo(CTX.getPackageName(),
0).versionName;
        } catch (PackageManager.NameNotFoundException e) {
            e.printStackTrace();
            return "";
        }
    }
```

这里就是获取版本号，我用的版本是：6.13.1

sessionId

```
clientApiReq.getHead().setSessionId(requestObject.getSessionid())
    public String getSessionid() {
        return this.sessionid;
    }


//但是在给sessionid赋值的只有这三个地方；两个构造器，一个set方法
```

```
public RequestObject() {
        this.key = null;
        this.sessionid = null;
    }

    public RequestObject(String str, String str2, boolean z) {
        this.key = str;
        this.sessionid = str2;
        this.islager = z;
    }

      public void setSessionId(String str) {
        this.sessionId = str;
    }
```

在desede加密的时候，`sessionId`的前24位是用作key的，每次重启app都会有不同的值

这里看了下`setSessionId`方法没被调用，`RequestObject()`也没被调用;

是通过`RequestObject(String str, String str2, boolean z)`来赋值的

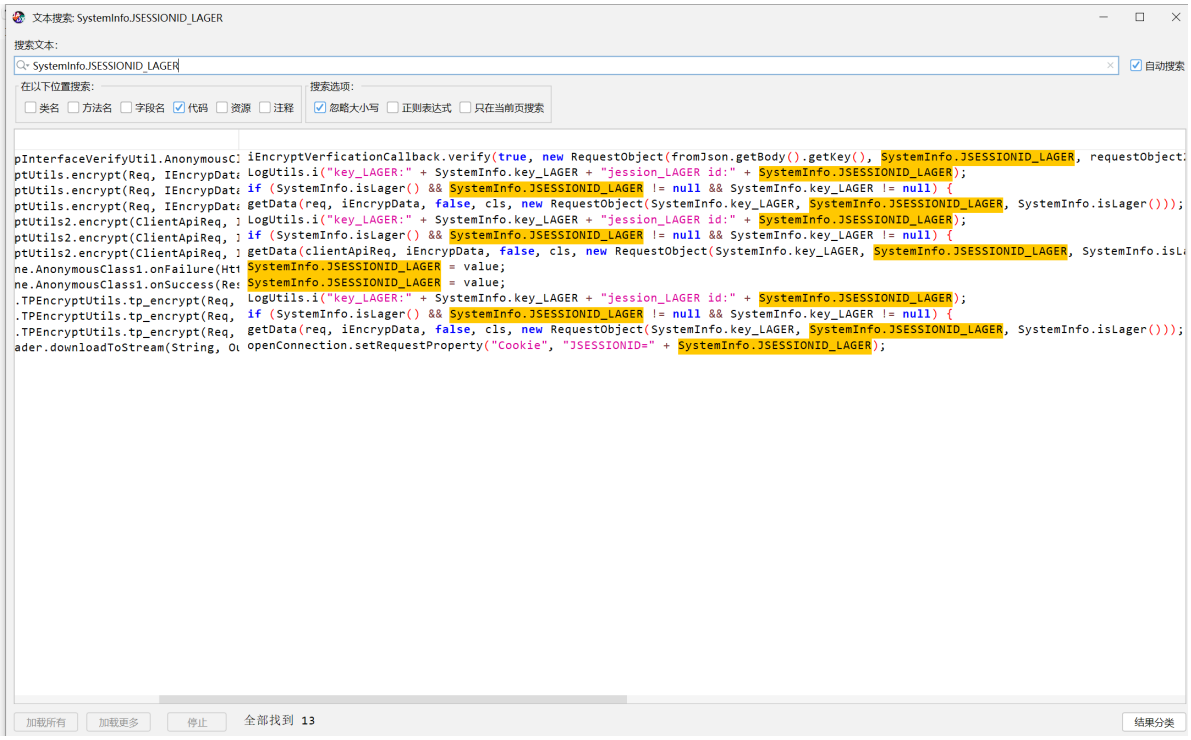
我们去打印一下`RequestObject(String str, String str2, boolean z)`的调用栈：

```
 RequestObject.$init is called:

str=GYCtoCcqX6IveHxPKEm38sPOW+lcygkzigz0/K6TSrRyW8OBg67hdGmlY39OtvrigOOmoo/Sv1F+
Mw+m7ArArjiH+O4Qo
5iV6LeqDrsur3OU7Z/RodZ2cRwtQHqlpwAS7+139fE3t4j7tXoWXSJyiujbIyxDlq0LeyR8IGk1Tr8=,


str2=E5BBD8AE25F889EDBCF2867A4E4AF559,

z=false

java.lang.Throwable

        at com.lidroid.xutils.http.RequestObject.<init>(Native Method)

        at
com.bw.zsch.client.android.service.encrypt.EncryptUtils2.encrypt(EncryptUtils2.j
ava:41)
        at
com.bw.zsch.client.android.service.ServiceDataImpl.callWeb2(ServiceDataImpl.java
:4444)
        at
com.bw.zsch.member.login.LoginActivity.lambda$doVerifyCodeLogin$16$com-bw-zsch-
member-login-LoginActivity(LoginActivity.java:493)
        at
com.bw.zsch.member.login.LoginActivity$$ExternalSyntheticLambda0.onToken(Unknown
Source:4)
        at com.bw.zsch.dxrisk.DXRiskUtils$1.run(DXRiskUtils.java:69)
```

可以看到是在EncryptUtils2.encrypt调用的这个构造方法

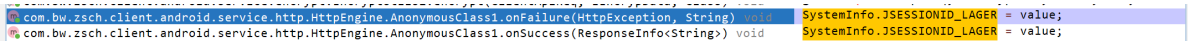
```
 getData(
     clientApiReq,
     iEncrypData,
     false,
     cls,
     new RequestObject(SystemInfo.key_LAGER, SystemInfo.JSESSIONID_LAGER,
SystemInfo.isLager())
 );
```

这里是在调用getData方法的时，在参数的位置，直接new了RequestObject然后把对应参数传了过去，SystemInfo.JSESSIONID_LAGER是我们的Sessionid，这是一个静态变量，并且看到是没有对应的set方法的。所以我们直接去搜类名.属性，看看能不能找到赋值的地方



搜到了13条，都去看看，有些是调用的看不看都行

只有倒数五、六条是赋值的，其他都是调用的，去看看那两条



这大概是在某个回调的时候去赋值的把，如果成功赋值，失败了也赋值

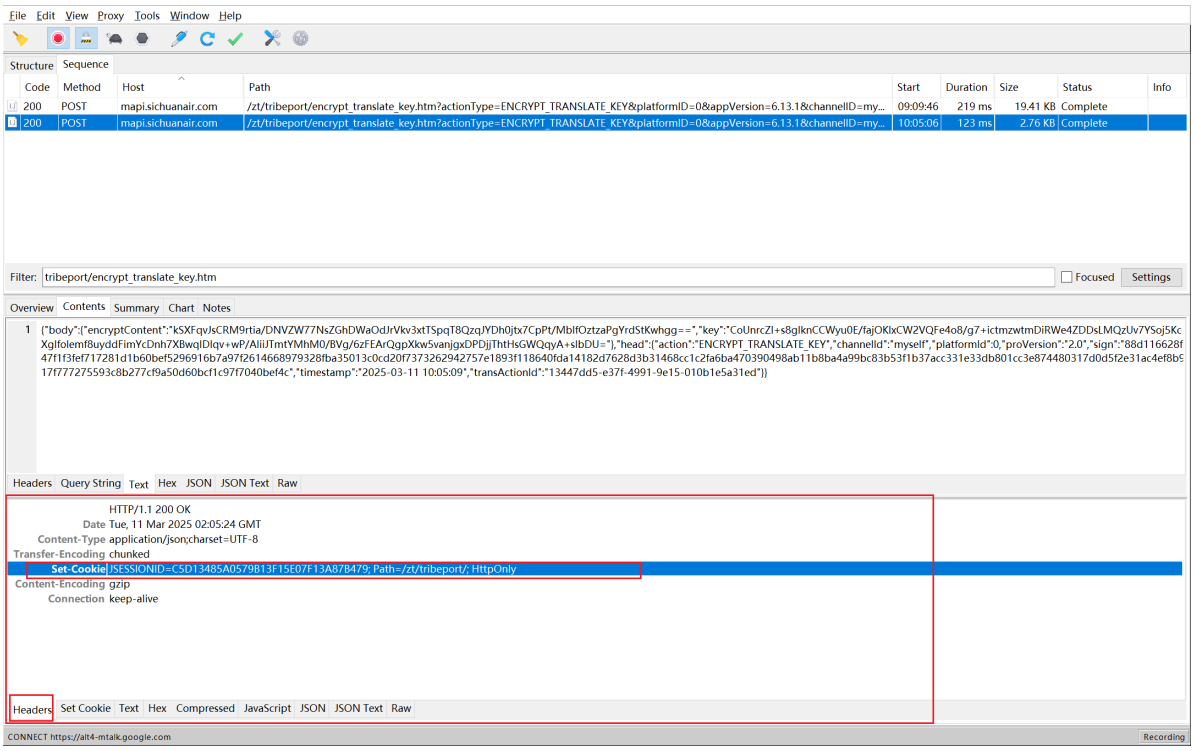
```
public static void post(String str, String str2, final IHttpCallback
iHttpCallback, RequestObject requestObject) {
        if (iHttpCallback != null) {
            hu = new HttpUtils();
            RequestParams requestParams = new RequestParams();
            try {
                requestParams.setBodyEntity(new StringEntity(str2));
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
            final CookieStore basicCookieStore = new BasicCookieStore();
            hu.configCookieStore(basicCookieStore);
```

```
            if (requestObject != null && requestObject.getSessionid() != null) {
                requestParams.setHeader("Cookie", "JSESSIONID=" +
requestObject.getSessionid());
            }
            mHttpHandler = hu.send(HttpRequest.HttpMethod.POST, str,
requestParams, new RequestCallBack<String>() {
                @Override
                public void onStart() {
                    super.onStart();
                    IHttpCallback.this.onStart();
                }

                @Override
                public void onLoading(long j, long j2, boolean z) {
                    super.onLoading(j, j2, z);
                    IHttpCallback.this.onLoading(j, j2, z);
                }

                @Override
                public void onFailure(HttpException httpException, String str3)
{
                    for (Cookie cookie : basicCookieStore.getCookies()) {
                        if (cookie.getName().equalsIgnoreCase("JSESSIONID")) {
                            LogUtils.i("cookie" + SystemInfo.JSESSIONID);
                            String value = cookie.getValue();
                            if (SystemInfo.isLager()) {
                                SystemInfo.JSESSIONID_LAGER = value;
                            } else {
                                SystemInfo.JSESSIONID = value;
                            }
                            LogUtils.i("cookie" + value);
                        }
                    }
                    IHttpCallback.this.onFailure(httpException, str3);
                }

                @Override
                public void onSuccess(ResponseInfo<String> responseInfo) {
                    for (Cookie cookie : basicCookieStore.getCookies()) {
                        LogUtils.i("cookie" + cookie.getName());
                        if (cookie.getName().equalsIgnoreCase("JSESSIONID")) {
                            LogUtils.i("cookie" + SystemInfo.JSESSIONID);

                            String value = cookie.getValue();
                            if (responseInfo.getRequestObject() != null &&
responseInfo.getRequestObject().islager) {

                                //设置sessionid
                                SystemInfo.JSESSIONID_LAGER = value;
                            } else {
                                SystemInfo.JSESSIONID = value;
                            }
                            LogUtils.i("cookie" + value);
                        }
                    }
```

```
                    LogUtils.i("返回:  " + responseInfo.result);
                    if (responseInfo.getRequestObject() != null &&
responseInfo.getRequestObject().getKey() != null) {
                        if (responseInfo.getRequestObject() != null &&
responseInfo.getRequestObject().islager) {
                            if (SystemInfo.key_LAGER != null &&
!SystemInfo.key_LAGER.equals(responseInfo.getRequestObject().getKey())) {
                                IHttpCallback.this.onSuccess(null, null);
                                return;
                            }
                        } else if (SystemInfo.key != null &&
!SystemInfo.key.equals(responseInfo.getRequestObject().getKey())) {
                            LogUtils.i("返回:  SystemInfo.key不一致");
                            IHttpCallback.this.onSuccess(null, null);
                            return;
                        }
                    }
                    IHttpCallback.this.onSuccess(responseInfo.result,
responseInfo.getRequestObject());
                }
            }, requestObject);
        }
    }
```

经过验证得到这个是从服务器拿到的，那既然是从服务器拿到的，就应该要发请求，然后下发把，我们去找找发请求的逻辑

我直接拿这个`sessionId`的值去搜索，在charles中找到了一条疑似的：



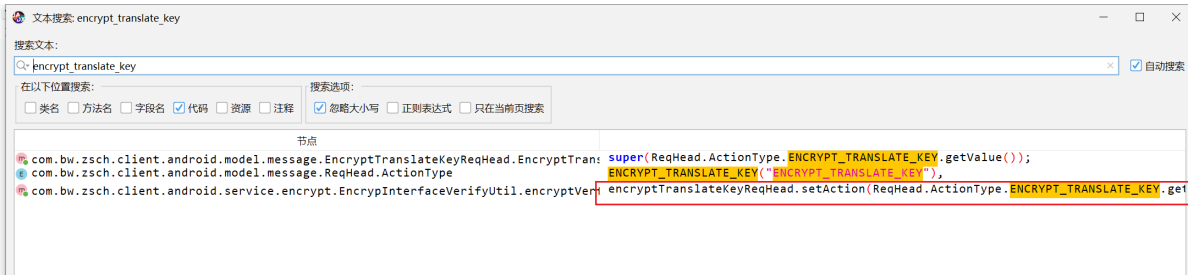
在这个响应头里，可以看到有这个sessionId的值

我们清除一下记录，重新打开app去看看；经过验证，就是从服务器下发的。并且多试了几次，只有打开app的时候会发这个请求，后边就不会再发了

向服务器索要`sessionId`请求

```
目标网址：https://mapi.sichuanair.com/zt/tribeport/encrypt_translate_key.htm
请求方式：POST
请求头：
    Content-Length  751
    Content-Type    text/plain; charset=ISO-8859-1
    Host    mapi.sichuanair.com
    Connection  Keep-Alive
    User-Agent  Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
    Accept-Encoding gzip
请求体：
    {
    "body": {
        "encryptContent":
"kSXFqvJsCRM9rtia/DNVZW77NsZGhDWaOdJrVkv3xtTSpqT8QzqJYDh0jtx7CpPt/MblfOztzaPgYrd
StKwhgg==",
        "key":
"CoUnrcZI+s8glknCCWyuOE/fajOKIxCW2VQFe4o8/g7+ictmzwtmDiRWe4ZDDsLMQzUv7YSoj5KcXgl
foIemf8uyddFimYcDnh7XBwqIDIqv+wP/AliiJTmtYMhM0/BVg/6zFEArQgpXkw5vanjgxDPDjjThtHs
GWQqyA+slbDU="
    },
    "head": {
        "action": "ENCRYPT_TRANSLATE_KEY",
        "channelId": "myself",
        "platformId": 0,
        "proVersion": "2.0",
        "sign":
"2e93b528dea6d2c517a993d645c2ee539754c8515aa2c049b4ef143893d2053b929b62a938be71c
de57a3d2eeb90074d08c202c943a3789fab5f34cde1fd723cad3cb0ac43183b32fb464bc6bd37c5e
f63d9bfc4effd08c29e561f5322b7fdd9fcc06a4399bb36ffe9f4132cd7319e4adf5c6514d304e84
e659c377458252558",
        "timestamp": "2025-03-11 18:32:24",
        "transActionId": "2f44daf9-3a3b-4d2e-99b9-473d901bd4ef"
    }
}
携带参数：
    actionType  ENCRYPT_TRANSLATE_KEY
    platformID  0
    appVersion  6.13.1
    channelID   myself
    imei    5a9992681fc4b10a
    networkOperator
    mac 02:00:00:00:00:00
    transActionId   2f44daf9-3a3b-4d2e-99b9-473d901bd4ef
```

我们去搜下这个链接



这个挺像，先去看看这个

```
public void encryptVerfication(final IEncryptVerficationCallback
iEncryptVerficationCallback, RequestObject requestObject) {
        try {
            String info = SystemInfo.getINFO();
            if (info != null && info.length() > 64) {
                info = info.substring(0, 63);
            }
            String encryptThreeDESECB = DESedeUtil.encryptThreeDESECB(info,
info, "UTF-8");
            System.loadLibrary("BWSecurity");
            String sign = BwSecurityNative.sign(info);
            EncryptTranslateKeyReqBody encryptTranslateKeyReqBody = new
EncryptTranslateKeyReqBody(sign, encryptThreeDESECB);
            EncryptTranslateKeyReqHead encryptTranslateKeyReqHead = new
EncryptTranslateKeyReqHead();
            encryptTranslateKeyReqHead.setPlatformId(0);

 encryptTranslateKeyReqHead.setAction(ReqHead.ActionType.ENCRYPT_TRANSLATE_KEY.g
etValue());
            encryptTranslateKeyReqHead.setProVersion(SystemInfo.PRO_VERSION);
            encryptTranslateKeyReqHead.setTimestamp(DateTool.getCurrentTime());
            encryptTranslateKeyReqHead.setChannelId(SystemInfo.getChannelID());

 encryptTranslateKeyReqHead.setTransActionId(UUID.randomUUID().toString());
            EncryptTranslateKeyReq encryptTranslateKeyReq = new
EncryptTranslateKeyReq(encryptTranslateKeyReqHead, encryptTranslateKeyReqBody);
            BwSecurityNative.reqSign(encryptTranslateKeyReq);
           //设置请求信息

            //调用post发送请求
            HttpEngine.post(
                SystemInfo.getRequestUrl(encryptTranslateKeyReqHead),
                encryptTranslateKeyReq.toJson(),
                new IHttpCallback() {//回调 },
                requestObject
                    );
        } catch (Exception e) {
            iEncryptVerficationCallback.verify(false, null);
            e.printStackTrace();
        }
    }
```

这里应该是调用的post发送的请求，我们去hook下看看post函数的参数：

```
HttpEngine.post is called:

str=https://mapi.sichuanair.com/zt/tribeport/encrypt_translate_key.htm?
actionType=ENCRYPT_TRANSLATE_KEY&platformID=0&appVersion=6.13.1&channelID=myself
&imei=5a9992681fc4b10a&networkOperator=&mac=02:00:00:00:00:00&transActionId=d24f
7151-237d-467e-86d2-33e567f28446,


str2={"body":
{"encryptContent":"kSXFqvJsCRM9rtia/DNVZW77NsZGhDWaOdJrVkv3xtTSpqT8QzqJYDh0jtx7C
pPt/MblfOztzaPgYrdStKwhgg==","key":"CoUnrcZI+s8glknCCWyu0E/fajOKIxCW2VQFe4o8/g7+
ictmzwtmDiRWe4ZDDsLMQzUv7YSoj5KcXglfoIemf8uyddFimYcDnh7XBwqIDIqv+wP/AliiJTmtYMhM
O/BVg/6zFEArQgpXkw5vanjgxDPDjjThtHsGWQqyA+slbDU="},"head":
{"action":"ENCRYPT_TRANSLATE_KEY","channelId":"myself","platformId":0,"proVersio
n":"2.0","sign":"8f75e044bc6014d4288b3395f59490eca36d50f860dedf9bdbddb3daef194cf
e360d818f3a7bff5d73ab4d12530c448e78a6bf8d7d20ed0e729e8ab97a6db788e3c401184bf3a4d
a856ce1ae2ba47fd7a56d1fcb49a6abe1c9e686210dc09619de32f06359b73c23e88d8bfe98e18d5
0cad4db5e190272b554433ee9c08a9476","timestamp":"2025-03-11
20:21:15","transActionId":"d24f7151-237d-467e-86d2-33e567f28446"}},

iHttpCallback=com.bw.zsch.client.android.service.encrypt.EncrypInterfaceVerifyUt
il$1@8e50f0b,

requestObject=RequestObject [key=null, sessionid=null, islager=false]




java.lang.Throwable
    at com.bw.zsch.client.android.service.http.HttpEngine.post(Native Method)
    at
com.bw.zsch.client.android.service.encrypt.EncrypInterfaceVerifyUtil.encryptVerf
ication(EncrypInterfaceVerifyUtil.java:66)
    at
com.bw.zsch.client.android.EncryptBeforeReqManager.moreRequest(EncryptBeforeReqM
anager.java:73)
    at
com.bw.zsch.client.android.service.encrypt.EncryptUtils.encrypt(EncryptUtils.jav
a:43)
    at
com.bw.zsch.client.android.service.ServiceDataImpl.tp_use_valid_system(ServiceDa
taImpl.java:4172)
    at
com.bw.zsch.member.login.LoginActivity.loadRiskTag(LoginActivity.java:161)
    at com.bw.zsch.member.login.LoginActivity.onCreate(Native Method)
    at android.app.Activity.performCreate(Activity.java:7802)
    at android.app.Activity.performCreate(Activity.java:7791)
    at
android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1306)
    at
android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3245)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3409)
    at
android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java
:83)
```

```
    at
android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionEx
ecutor.java:135)
    at
android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.ja
va:95)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2016)
    at android.os.Handler.dispatchMessage(Handler.java:107)
    at android.os.Looper.loop(Looper.java:214)
    at android.app.ActivityThread.main(ActivityThread.java:7356)
    at java.lang.reflect.Method.invoke(Native Method)
    at
com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:492
)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:930)
```

```
public void encryptVerfication(final IEncryptVerficationCallback
iEncryptVerficationCallback, RequestObject requestObject) {
        try {
            String info = SystemInfo.getINFO();
            if (info != null && info.length() > 64) {
                info = info.substring(0, 63);
            }
            String encryptThreeDESECB = DESedeUtil.encryptThreeDESECB(info,
info, "UTF-8");
            System.loadLibrary("BWSecurity");
            String sign = BwSecurityNative.sign(info);
            EncryptTranslateKeyReqBody encryptTranslateKeyReqBody = new
EncryptTranslateKeyReqBody(sign, encryptThreeDESECB);
            EncryptTranslateKeyReqHead encryptTranslateKeyReqHead = new
EncryptTranslateKeyReqHead();
            encryptTranslateKeyReqHead.setPlatformId(0);

 encryptTranslateKeyReqHead.setAction(ReqHead.ActionType.ENCRYPT_TRANSLATE_KEY.g
etValue());
            encryptTranslateKeyReqHead.setProVersion(SystemInfo.PRO_VERSION);
            encryptTranslateKeyReqHead.setTimestamp(DateTool.getCurrentTime());
            encryptTranslateKeyReqHead.setChannelId(SystemInfo.getChannelID());

 encryptTranslateKeyReqHead.setTransActionId(UUID.randomUUID().toString());
            EncryptTranslateKeyReq encryptTranslateKeyReq = new
EncryptTranslateKeyReq(encryptTranslateKeyReqHead, encryptTranslateKeyReqBody);
            BwSecurityNative.reqSign(encryptTranslateKeyReq);
           //设置请求信息

            //调用post发送请求，经过hook得知，post的第二个参数，即
encryptTranslateKeyReq就是我们的请求体
            //请求参数也在encryptTranslateKeyReq里边，
            //是对应的encryptTranslateKeyReq里边的encryptTranslateKeyReqHead
            HttpEngine.post(
                SystemInfo.getRequestUrl(encryptTranslateKeyReqHead),
                encryptTranslateKeyReq.toJson(),
                new IHttpCallback() {//回调逻辑},
```

```
                requestObject);
        } catch (Exception e) {
            iEncryptVerficationCallback.verify(false, null);
            e.printStackTrace();
        }
    }
```

请求数据的组成：请求头，请求体，请求参数

关于请求体：

```
请求体是由两部分组成的：encryptTranslateKeyReqBody、encryptTranslateKeyReqHead

    //调用post发送请求，经过hook得知，post的第二个参数，即encryptTranslateKeyReq就是我们的请求体
    //我们去看看encryptTranslateKeyReq是怎么来的，往上看，可以看到：
    EncryptTranslateKeyReqBody encryptTranslateKeyReqBody = new EncryptTranslateKeyReqBody(sign, encryptThreeDESECB);
    EncryptTranslateKeyReqHead encryptTranslateKeyReqHead = new EncryptTranslateKeyReqHead();

    EncryptTranslateKeyReq encryptTranslateKeyReq = new EncryptTranslateKeyReq(encryptTranslateKeyReqHead, encryptTranslateKeyReqBody);


可以看到是先new了encryptTranslateKeyReqHead和encryptTranslateKeyReqBody，然后再去new encryptTranslateKeyReq，然后把他们两个传进去。他们就分别对应的是EncryptTranslateKeyReq中的head和body。对于body是传参数然后直接new出来的;head是先new了一个空的，然后再去设置的参数

-----------------------------------------------分割线-------------------------------------------------

关于body：
        String encryptThreeDESECB = DESedeUtil.encryptThreeDESECB(info, info, "UTF-8");
        String sign = BwSecurityNative.sign(info);
        encryptTranslateKeyReqBody = new EncryptTranslateKeyReqBody(sign, encryptThreeDESECB);

去看看EncryptTranslateKeyReqBody的构造函数：
             public EncryptTranslateKeyReqBody(String str, String str2) {
                                this.key = str;
                                this.encryptContent = str2;
                        }
        第一个参数就是body里的key，第二个参数就body里的encryptContent
        "body": {
                "encryptContent": "kSXFqvJsCRM9rtia/DNVZW77NsZGhDWaOdJrVkv3xtTSpqT8QzqJYDhOjtx7CpPt/MblfOztzaPgYrdStKwhgg==",
                "key": "CoUnrcZI+s8glknCCWyuOE/fajOKIxCW2VQFe4o8/g7+ictmzwtmDiRWe4ZDDsLMQzUv7YSoj5KcXglfoIemf8uyddFimYcDnh7XBwqIDIqv+wP/AliiJTmtYMhMO/BVg/6zFEArQgpXkw5vanjgxDPDjjThtHsGWQqyA+slbDU="
            },
```

我们再去单独看他们生成的逻辑，先看第二个参数：encryptContent，因为这个是java层的，简单一些：

```
String info = SystemInfo.getINFO();
if (info != null && info.length() > 64) {
    info = info.substring(0, 63);
}
String encryptThreeDESECB = DESedeUtil.encryptThreeDESECB(info, info, "UTF-8");
```

首先是获取info，然后判断如果info不为空并且长度大于64，那么就取前64位

这里我们去看看SystemInfo.getINFO()

```
public static String getINFO() {
    if (getPlatFormId().intValue() == 0) {
        return "&platformID=" + getPlatFormId() + "&appVersion=" + getAppVersion() + "&channelID=" + getChannelID() + "&imei=" + getImei() + "&networkOperator=" + getSimOperatorName() + "&mac=" + getMacAddress();
    }
    return "&platformID=" + getPlatFormId() + "&appVersion=" + getApp_Version() + "&channelID=" + getChannelID() + "&imei=" + getImei() + "&networkOperator=" + getSimOperatorName() + "&mac=" + getMacAddress();
}
```

我们可以去hook一下这个getINFO，看看返回的是什么：
&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681fc4b10a&networkOperator=&mac=02:00:00:00:00:00

我们取前64位就是：
&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681

接下来就是调用DESedeUtil.encryptThreeDESECB(info, info, "UTF-8")然后拿到encryptThreeDESECB

我们去看看DESedeUtil.encryptThreeDESECB

```
public static String encryptThreeDESECB(String str, String str2, String str3) throws Exception {
    SecretKey generateSecret = SecretKeyFactory.getInstance("DESede").generateSecret(new DESedeKeySpec(str2.getBytes(str3)));
    Cipher cipher = Cipher.getInstance("desede/CBC/PKCS5Padding");
    cipher.init(1, generateSecret, new IvParameterSpec(iv.getBytes()));
    return Base64.encode(cipher.doFinal(str.getBytes(str3)));
}
```

这里是做了一个desede/CBC/PKCS5Padding加密，key为：传进去的info，iv为："01234567"，最后进行base64编码，就得到了encryptThreeDESECB。 这里key是24位的，所以会截取前24位：
&platformID=0&appVersion

这里可以固定写死，或者一会变化着去看看，应该都可以发请求。

现在去看看body里的key，也就是第一个参数：

```
String sign = BwSecurityNative.sign(info);
encryptTranslateKeyReqBody = new EncryptTranslateKeyReqBody(sign, encryptThreeDESECB);
```

这个info和之前的是同一个：
&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681

然后通过BwSecurityNative.sign(info)得到的sign

现在去看看这个 BwSecurityNative.sign

```
int __fastcall Java_com_bw30_zsch_security_BwSecurityNative_sign(JNIEnv *env, int a2, int a3){
                int v3; // r0
                void *v5; // [sp+4h] [bp-1Ch]
                _BYTE *info_cstr; // [sp+8h] [bp-18h]

                info_cstr = j_jstringTostring(env info);
                v5 = j_stoJbyteArray(env, info_cstr);
                v3 = j_rsa_encrpy_by_public_key(env, v5);
                return j_base64_encode(env, v3);
            }
```

这里是进行了一个rsa加密，然后转base64.公钥为：

MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+QkdMYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB

到这里请求体中的 body部分就解决了：

info：&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681

body中的key：通过rsa对info进行加密后转为base64编码

body中的encryptContent：desede/CBC/PKCS5Padding加密后转为base64编码，

其中key为：&platformID=0&appVersion，

iv为：01234567

----------分割线----------

关于head：


```
    "head": {
            "action": "ENCRYPT_TRANSLATE_KEY",
            "channelId": "myself",
            "platformId": 0,
            "proVersion": "2.0",
            "sign":
"2e93b528dea6d2c517a993d645c2ee539754c8515aa2c049b4ef143893d2053b929b62a938be71cde57a3d2eeb90074d08c202c943a3789fab5f34cde1fd723cad3cb0ac43183b32fb464bc6bd37c5ef63d9bfc4effd08c29e561f5322b7fdd9fcc06a4399bb36ffe9f4132cd7319e4adf5c6514d304e84e659c377458252558",
            "timestamp": "2025-03-11 18:32:24",
            "transActionId": "2f44daf9-3a3b-4d2e-99b9-473d901bd4ef"
        }
```


主要有这么几个参数需要搞定：action、channelId、platformId、proVersion、sign、timestamp、transActionId

我们先去看看new EncryptTranslateKeyReqHead()那里，new的时候是空参构造，new完后用set设置了对应值：

```
encryptTranslateKeyReqHead.setPlatformId(0);

 encryptTranslateKeyReqHead.setAction(ReqHead.ActionType.ENCRYPT_TRANSLATE_KEY.getValue());
encryptTranslateKeyReqHead.setProVersion(SystemInfo.PRO_VERSION);
```

```
            encryptTranslateKeyReqHead.setTimestamp(DateTool.getCurrentTime());
            encryptTranslateKeyReqHead.setChannelId(SystemInfo.getChannelID());

 encryptTranslateKeyReqHead.setTransActionId(UUID.randomUUID().toString());
            BwSecurityNative.reqSign(encryptTranslateKeyReq);
```

platformId:

```
    encryptTranslateKeyReqHead.setPlatformId(0);
```

这里直接赋值0

action:

```
encryptTranslateKeyReqHead.setAction(ReqHead.ActionType.ENCRYPT_TRANSLATE_KEY.ge
tValue());
```

这个是获取枚举变量的值，这里对应的就是：ENCRYPT_TRANSLATE_KEY

proVersion:

```
encryptTranslateKeyReqHead.setProVersion(SystemInfo.PRO_VERSION);
public static String PRO_VERSION = "2.0";
```

这里就是：2.0

Timestamp:

```
encryptTranslateKeyReqHead.setTimestamp(DateTool.getCurrentTime());

    public static String getCurrentTime() {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat();
        Date date = new Date();
        simpleDateFormat.applyPattern("yyyy-MM-dd HH:mm:ss");
        return simpleDateFormat.format(date);
    }
```

时间戳是通过Data和SimpleDateFormat类得到的，按照"yyyy-MM-dd HH:mm:ss"格式输出

channelId:

```
encryptTranslateKeyReqHead.setChannelId(SystemInfo.getChannelID());

    public static String getChannelID() {
        Object obj;
        try {
            ApplicationInfo applicationInfo =
CTX.getPackageManager().getApplicationInfo(CTX.getPackageName(), 128);
            return (applicationInfo.metaData == null || (obj =
applicationInfo.metaData.get("ZSCH")) == null) ? "" : obj.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }
```

这里是判断app是通过什么渠道下载的，这里是同adb安装的，所以这里的值为：myself

transActionId：
  encryptTranslateKeyReqHead.setTransActionId(UUID.randomUUID().toString());
  这里就是直接拿一个uuid

 sign：
      BwSecurityNative.reqSign(encryptTranslateKeyReq);
      这里是设置sign的，我们之前分析native方法encrypt的时候就遇到过签名的sign，那个是拼接字符串，然后md5=>hex=>RSa=>hex
      我们去reqSign函数中看看，是什么逻辑：

```
int __fastcall Java_com_bw30_zsch_security_BwSecurityNative_reqSign(JNIEnv *env, jclass jclass, void *req){
                        int v3; // r0
                        void *v5; // [sp+10h] [bp-28h]
                        void *v6; // [sp+14h] [bp-24h]
                        void *v7; // [sp+18h] [bp-20h]
                        void *v8; // [sp+1Ch] [bp-1Ch]
                        void *v9; // [sp+20h] [bp-18h]

                        v9 = j_req_sign(env, req);
                        v8 = j_md5_encrypt(env, v9);
                        v7 = j_hex_encode(env, v8);
                        v6 = j_jstringToJbyteArray(env, v7);
                        v5 = j_rsa_encrpy_by_public_key(env, v6);
                        v3 = j_hex_encode(env, v5);
                        return j_set_req_sign(env, req, v3);
                }
```

    跟之前的是共用的同一批函数，整体逻辑如下：
        拼接字符串=>对拼接字符串进行MD5加密=>对MD5结果进行hex编码=>RSA加密=>RSA密文进行hex编码=>设置到head中的sign里

    2025-03-0631743738-5a2f-4a09-8f83-4f85359248102822563731

    2025-03-071584b7ee-3039-43b7-b9f0-cbc7925e3f142822563731

第一部分	第二部分	第三部分
2025-03-07	1584b7ee-3039-43b7-b9f0-cbc7925e3f14	2822563731

    第一部分：timestamp的年月日部分
    第二部分：transActionId
    第三部分：dex文件CRC值拼接起来：2822563731

经过验证，的确是和之前的sign是相同的算法

```
head的总结：action、channelId、platformId、proVersion、sign、timestamp、transActionId
        action固定值："ENCRYPT_TRANSLATE_KEY"
        channelId固定值："myself"
        platformId固定值：0
        proVersion固定值(不换版本就不需要换)："2.0"
        timestamp：通过Data和SimpleDateFormat类得到的，按照"yyyy-MM-dd HH:mm:ss"格式输出
        transActionId ：UUID
        sign：timestamp的年月日部分+transActionId+dex文件CRC值拼接起来，然后对拼接字符串进行MD5加密=>对MD5结果进行hex编码=>RSA加密=>RSA密文进行hex编码=>结果
        每个版本的dex大小不同，得到的CRC的值应该也不同，6.13.1版本的值为：2822563731
        CRC主要是校验dex文件的完整性的，类似于哈希
 ------------------------------------------------分割线-----------------------------------------------------
```

请求参数：

```
我们在对post进行hook的时候，发现post有四个参数，其中第二个是我们的请求体，但是我们当时没看第一个，我们把第一个参数拿过来看看：
https://mapi.sichuanair.com/zt/tribeport/encrypt_translate_key.htm?
actionType=ENCRYPT_TRANSLATE_KEY&
platformID=0&
appVersion=6.13.1&
channelID=myself&
imei=5a9992681fc4b10a&networkOperator=&
mac=02:00:00:00:00:00&
transActionId=d24f7151-237d-467e-86d2-33e567f28446,
对比抓包得到的参数：
    actionType  ENCRYPT_TRANSLATE_KEY
    platformID  0
    appVersion  6.13.1
    channelID   myself
    imei    5a9992681fc4b10a
    networkOperator
    mac 02:00:00:00:00:00
    transActionId   2f44daf9-3a3b-4d2e-99b9-473d901bd4ef
 可以发现，psot的第一个参数就是我们的请求参数，不过他给组装起来了，我们在使用request发送请求的时候，自己组装起来像参数1这样也行，或者自己写一个表单放进去。

 所以我们去追这个参数1的来历，这样就能搞明白这些参数是怎么来的了：
        SystemInfo.getRequestUrl(encryptTranslateKeyReqHead)；
        它是把encryptTranslateKeyReqHead传进去拼接的：
   public static String getRequestUrl(ReqHead reqHead) {
        StringBuilder sb = new StringBuilder();
        sb.append("https://mapi.sichuanair.com/zt/tribeport/");
        sb.append(TextUtils.isEmpty(reqHead.getPath()) ? "" : reqHead.getPath());//空

 sb.append(reqHead.getAction().toLowerCase(Locale.CHINA));//encrypt_translate_key
        sb.append(".htm?actionType=");
```

```
        sb.append(TextUtils.isEmpty(reqHead.getCallWebAction()) ?
reqHead.getAction() :
reqHead.getCallWebAction());//ENCRYPT_TRANSLATE_KEY
        sb.append(getINFO());
        sb.append("&transActionId=");
        sb.append(reqHead.getTransActionId());
        return sb.toString();
    }

getINFO():
public static String getINFO() {
        if (getPlatFormId().intValue() == 0) {
            return "&platformID=" + getPlatFormId() + "&appVersion=" +
getAppVersion() + "&channelID=" + getChannelID() + "&imei=" + getImei() +
"&networkOperator=" + getSimOperatorName() + "&mac=" + getMacAddress();
        }
        return "&platformID=" + getPlatFormId() + "&appVersion=" +
getApp_Version() + "&channelID=" + getChannelID() + "&imei=" + getImei() +
"&networkOperator=" + getSimOperatorName() + "&mac=" + getMacAddress();
    }

getINFO()返回的是:
&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681fc4b10a&networkOp
erator=&mac=02:00:00:00:00:00

总的说就那几个参数都是在head里的。只有个别几个是不在的，并且那些也是固定的
    actionType同head: ENCRYPT_TRANSLATE_KEY
    platformID同head: 0
    appVersion同head: 6.13.1
    channelID同head: myself
    imei固定的(跟设备信息有关):5a9992681fc4b10a
    networkOperator(固定的): 空
    mac(固定的): 02:00:00:00:00:00
    transActionId同head: 2f44daf9-3a3b-4d2e-99b9-473d901bd4ef

 所以这个请求参数，大部分是和请求体的head一样，有个别的是固定的
```

请求头:

请求头看着没有什么加密的东西，可以直接使用看看

到这里，向服务器发请求拿sessionId就结束了，我们去自己发请求拿一下看看

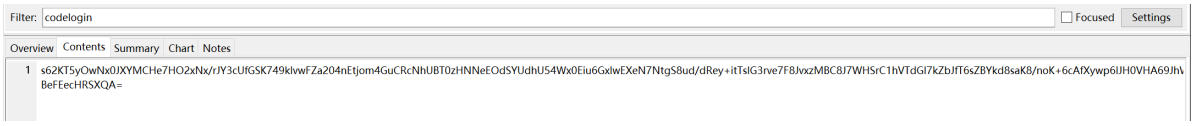
customerId

```
clientApiReq.getHead().setCustomerId(SystemInfo.customerId);

public static String customerId = "";
```

这个默认是空，在进行设置请求头的时候也是空

请求体

我们看到，请求体是加密了的



之前分析过，它是进行了desede/CBC/PKCS5Padding加密然后base64编码的，其key为sessionId的前24位，iv：01234567

所以我们拿到响应体后可以先进行base64解码，然后desede/CBC/PKCS5Padding解密，然后再转String就可以看到响应体了。

hook-java层算法通杀

```
----------加密----------
 desede/CBC/PKCS5Padding init key  Utf8:  0C9AE2425AD050460C7A910A
-------------------------------------------------------------
 desede/CBC/PKCS5Padding init IV  Utf8:  01234567
=============================================================
Cipher.doFinal.overload('[B') is called!

 desede/CBC/PKCS5Padding doFinal data  Utf8:  {"body":
{"mobileCountryCode":"86","phone":"15512103215","verifyCode":"123456"},"head":
{"action":"clientapi/login/member/codeLogin","appVersion":"6.13.1","channelId":"
myself","customerId":"","imie":"00000000-6fd2-5a75-ffff-
ffffca01fdf4","jpushId":"120c83f7611feca6238","macAddress":"02:00:00:00:00:00","
platformId":0,"proVersion":"2.0","sessionId":"0C9AE2425AD050460C7A910AE682BFEA",
"sign":"6f1ca25394a3e4b40b8c399d202efaf682f914a4cf73137f496d5f4ad5fab3ef9a3a5061
9641a801886f7424841e8abda86a2fc68edaa2fb4433d12454a18a2bafe8f4d8ac2e975f4c39181a
2d286009c1c028b238c44567f0ede4f218f15589d1dc740993a1e5a59054fb5901722ac8ddeeca62
ff90fc6552513b8df80cb82f","timestamp":"2025-03-09
16:43:13","transActionId":"0cbcb77b-6bf2-46ec-ae97-5cbb7b61fe37"},"riskReqBody":
{"constId":"67cd54a1h118yuhL38dC82jSn8w0GGc3EbwSZ3n3","validType":"0"}}
-------------------------------------------------------------
 desede/CBC/PKCS5Padding doFinal result  Base64:
uJ/B4Y3fDwHwnM3xXntIgM0Fh9IfC9vogRZkB8mYyCVSJ3X4qOOTkUPeKG+j6MC5t7xqBI68wwW4iJYB
8U+mUHbOacO6Pcs/+Dds3zsOc+thsPyNhOsaljwSALOOzyphWZ/eFId3O3EqTDxaCsX1jxCDCblFcGTQ
dVqnyMxlr1uapkJGwt+sG7csOK4dmpgIzhrqszpAmBqJ+G7QLF6wtFeoUspQ8TFmknm/+NOARslM2+/g
UaWYGvkmhgTRpMXGh562yHRHsKn67RlVP9Ge64wJOfj6tHEPDTMO1iGsca/Boyio15+DJJ3sUibOoeQI
+HZ+q5WZ4Yc2j+bP3bDLt7O7XIDtpBVIXagYhpHy8y3uRqsD53ZvH687tQlg1yYPMSARNXHjU99kIlBF
qWO14V7hUtRmv4WCBSTKasawMBdDU75ViqFpu4g4dEptO23ZeUS8GOWPG7LfFs3rBl89qosoavyX3vFN
+qSW6iuIGLxmJML3IFX5kN/ASdrhFj+MfXx5umlbf3KMVUOXJMtMS4MvLmic3G52FKuMzYtvIEcTb6fH
CMqrc+xUfmTpq2OVStexHqnCDHJoUCCbrK+I7nRkzet38lahQLhiSi693f6DHrncnwa7pf7NADNF4Oj0
r5vCcrHaV7/Tu3NkIfqhJWyFqlirkpwwk2oIHYtBmu7xMTjlBG2hKvGwBCE7bn+saJRzJ7oQFcbr4bDQ
rH65M+z3E8Heb9pBUPAowzp2/pGWXCLcmEpPOzRqtqDcl6RPePltPekCkxrYgvF5PJDvwOQ3DSXKRMbP
hjAcY3poVt4mHThdOx3aZif7PlhLFV5wO9v+vFhLUKXmz6g1IX8EenH1jsAd78PJ+mU5egKEoH7x8HvH
O1IoXO65KNGT38/QLpJc+/cjJ5D+AdbUWrj2tLKEUt6hT9kRay5YNCGrfQVZ/jyvSsPHlzPcT/3T9HUL
yJDrxFocshOg9KXSD5vfbFpri5o+hvrgfUsXHsxQgsLRXtahjGIBpci+v4dxzcjNDxd07cbv9ynSs6sP
OFy+r2ii6gfg3MSPKTEJTMPYAzlhWGXiOeH/3nyutPnDBOgrdRUbda6aVoM=
=============================================================
```

我们可以看到这个加密前的请求体分为三部分：body、head、riskReqBody


```
{
"body":{"mobileCountryCode":"86","phone":"15512103215","verifyCode":"123456"},

"head":{
"action":"clientapi/login/member/codeLogin",
"appVersion":"6.13.1",
"channelId":"myself",
"customerId":"",
"imie":"00000000-6fd2-5a75-ffff-ffffca01fdf4",
"jpushId":"120c83f7611feca6238",
"macAddress":"02:00:00:00:00:00",
"platformId":0,
"proVersion":"2.0",
"sessionId":"0C9AE2425AD05046OC7A910AE682BFEA",
"sign":"6f1ca25394a3e4b40b8c399d202efaf682f914a4cf73137f496d5f4ad5fab3ef9a3a5061
9641a801886f7424841e8abda86a2fc68edaa2fb4433d12454a18a2bafe8f4d8ac2e975f4c39181a
2d286009c1c028b238c44567f0ede4f218f15589d1dc740993a1e5a59054fb5901722ac8ddeeca62
ff90fc6552513b8df80cb82f",
"timestamp":"2025-03-09 16:43:13",
"transActionId":"0cbcb77b-6bf2-46ec-ae97-5cbb7b61fe37"
},


"riskReqBody":
{"constId":"67cd54a1h118yuhL38dC82jSn8w0GGc3EbwSZ3n3","validType":"0"}

}
```


head的参数咱们已经分析过了

下边就是把body和riskReqBody搞定即可

body的话不用分析，直接就是手机号，验证码，国际区号

看看riskReqBody，跟着调用栈往前看把

```
    public static void encrypt(ClientApiReq clientApiReq, IEncrypData
iEncrypData, Class cls) {
        if (SystemInfo.isLager() && SystemInfo.JSESSIONID_LAGER != null &&
SystemInfo.key_LAGER != null) {
            getData(clientApiReq, iEncrypData, false, cls, new
RequestObject(SystemInfo.key_LAGER, SystemInfo.JSESSIONID_LAGER,
SystemInfo.isLager()));
        } else if (!SystemInfo.isLager() && SystemInfo.JSESSIONID != null &&
SystemInfo.key != null) {
            getData(clientApiReq, iEncrypData, false, cls, new
RequestObject(SystemInfo.key, SystemInfo.JSESSIONID, SystemInfo.isLager()));
        } else {
            EncryptBeforeReqManager.moreRequest(clientApiReq, iEncrypData,
false, cls, SystemInfo.isLager(), 2);
        }
    }
```

可以看到是在encypt中调用的getData，在这里没看到设置clientApiReq，我们再去hook一下，看看在encrypt的时候，clientApiReq都有什么

```
╠═ riskReqBody=TPSelfChannelRiskControllerSystemReqBody {
║   ├─ validType='0'
║   ├─ funcType='null'
║   ├─ constId='67cd7d62QGzGOPUd58ccv4RXOQawlfiHqw52oW23'
║   ├─ token='null'
║   ├─ ip='null'
║   ├─ phoneNumber='null'
║   ├─ userId='null'
║   ├─ userName='null'
║   ├─ source='null'
║   ├─ extCurrentUrl='null'
║   ├─ extCookie='null'
║   ├─ extSessionId='null'
║   ├─ extAccountExist='null'
║   ├─ extPasswordCorrect='null'
║   ├─ extIdState='null'
║   ├─ extLoginType='null'
║   ├─ extId='null'
║   ├─ extIdNo='null'
║   ├─ email='null'
║   ├─ memberLevel=null
║   ├─ serchFlightRequest=null
║   ├─ submitOrderRequest=null
║   ├─ activityRequest=null
║   ├─ payRequest=null
║   └─ refundTicketRequest=null
║ }
}


}
```

这里有个多态的关系，它传过来的不是clientApiReq的对象，而是clientApiReq子类：ClientApiRiskReq的对象，所以再打印的时候才会有这么多信息

```
public class ClientApiRiskReq<T, R> extends ClientApiReq<T> implements
Serializable {
    private static final long serialVersionUID = 1443572875916929462L;
    @NotNull(message = "风控参数不能为空")
    @ApiModelProperty("风控参数")
    private R riskReqBody;

    public R getRiskReqBody() {
        return this.riskReqBody;
    }

    public void setRiskReqBody(R r) {
        this.riskReqBody = r;
    }

    @Override
    public String toString() {
        return "ClientApiRiskReq{head=" + this.head + ", body=" + this.body + ",
riskReqBody=" + this.riskReqBody + "} " + super.toString();
    }
}
```

所以在保存请求信息的时候，一直是保存到了ClientApiRiskReq类的对象里了

回来继续看，在这个encrypt函数的时候，riskReqBody里的constId就已经有值了，继续往前追。

callWeb2

```
    public void callWeb2(Object obj, Object obj2, String str, final
ServiceDataCallback<ClientApiResult> serviceDataCallback) {
        ClientApiRiskReq clientApiRiskReq = new ClientApiRiskReq();
        clientApiRiskReq.setBody(obj);
        clientApiRiskReq.setRiskReqBody(obj2);
        clientApiRiskReq.setHead(new ReqHead(str));
        //设置 Body、RiskReqBody、Head

        EncryptUtils2.encrypt(clientApiRiskReq, new IEncrypData() {
            //调用encrypt，传进去了clientApiRiskReq
            @Override // com.bw.zsch.client.android.IEncrypData
            public void encryptData(Object obj3) {
                ServiceDataCallback serviceDataCallback2 = serviceDataCallback;
                if (serviceDataCallback2 == null || obj3 == null) {
                    return;
                }
                serviceDataCallback2.data((ClientApiResult) obj3);
            }
        }, ClientApiResult.class);
    }
```

可以看到是在这里进行`new ClientApiRiskReq();`的，并且设置了 `Body、RiskReqBody、Head`。

参数1给了body、参数2给了RiskReqBody、参数3是new ReqHead(str)后给了Head，追进去后可以看到就是把参数3给了head 的action

我们重点去看这个参数2

我们去hook一下这个callWeb2，看看传进来的参数是什么

```
ServiceDataImpl.callWeb2 is called:
参数1：CodeLoginReqBody(phone=15512103215, verifyCode=123456, picCode=null,
mobileCountryCode=86),

参数2：TPSelfChannelRiskControllerSystemReqBody{validType='0', funcType='null',
constId='67cd8b18VkfopnV9GBtT8mTCU4byx2SO5jMYnoH3', token='null', ip='null',
phoneNumber='null', userId='null', userName='null', source='null',
extCurrentUrl='null', extCookie='null', extSessionId='null',
extAccountExist='null', extPasswordCorrect='null', extIdState='null',
extLoginType='null', extId='null', extIdNo='null', email='null',
memberLevel=null, serchFlightRequest=null, submitOrderRequest=null,
activityRequest=null, payRequest=null, refundTicketRequest=null},

参数3：clientapi/login/member/codeLogin,

参数4：[object Object]
```

```
在这个时候，validType和constId就已经有值了
```

继续往前追，去看看com.bw.zsch.member.login.LoginActivity.lambda$doVerifyCodeLogin$15$com-bw-zsch-member-login-LoginActivity..最终找到了m514xc4c291da

```
 public void m514xc4c291da(final CodeLoginReqBody codeLoginReqBody, String str)
{
        final TPSelfChannelRiskControllerSystemReqBody
tPSelfChannelRiskControllerSystemReqBody = new
TPSelfChannelRiskControllerSystemReqBody();
        tPSelfChannelRiskControllerSystemReqBody.setValidType("0");
        tPSelfChannelRiskControllerSystemReqBody.setConstId(str);
        //设置RiskReqBody的参数


        ServiceFactory.getBasciFactory().callWeb2(
            codeLoginReqBody,
            tPSelfChannelRiskControllerSystemReqBody,
            "clientapi/login/member/codeLogin",
            new ServiceDataCallback() {
            @Override
```

```
            public final void data(Object obj) {
                LoginActivity.this.m513x6da4a0fb(
                    tPSelfChannelRiskControllerSystemReqBody,
                    codeLoginReqBody,
                    (ClientApiResult) obj
                );
            }
        });
    }
/*
这里是通过ServiceFactory.getBasciFactory().callWeb2发送请求
参数1 是验证码登录的请求体
参数2 是封装了风险控制相关信息
参数3 是目标urlz
参数4 是回调函数，在请求成功后，进行调用data函数.在data方法中，将请求体对象
tPSelfChannelRiskControllerSystemReqBody、codeLoginReqBody和返回的结果对象（转换为
ClientApiResult类型）作为参数，调用LoginActivity类中的m513x6da4a0fb方法，以进一步处理登录
请求的结果，例如判断登录是否成功，处理可能的风险验证等情况。
*/
```

可以看到这个constId值是在调用m514xc4c291da时传进来的，我们需要继续往前追

追到了doVerifyCodeLogin中

```
 private void doVerifyCodeLogin(String str, String str2) {
        final CodeLoginReqBody codeLoginReqBody = new CodeLoginReqBody();
        codeLoginReqBody.setPhone(str);
        codeLoginReqBody.setVerifyCode(str2);

 codeLoginReqBody.setMobileCountryCode(this.mBinding.getMobileCountryCode().subs
tring(1));
        CommonUtils.cleanTokenid(this);
        showPd();
        this.mRespUtils.setLoginType(1);
        DXRiskUtils.getToken(new DXRiskUtils.TokenCallback() {
            @Override
            public final void onToken(String str3) {
                LoginActivity.this.m514xc4c291da(codeLoginReqBody, str3);
            }
        });
    }
```

可以看到在调用LoginActivity.this.m514xc4c291da(codeLoginReqBody, str3);时是把str3传进去了

我们去看看这个onToken的调用

跟过来后发现，这个constId就是Token，是通过getToken获得的：

```
public static void getToken(final TokenCallback tokenCallback) {
        new Thread() {
            @Override // java.lang.Thread, java.lang.Runnable
            public void run() {
                HashMap hashMap = new HashMap();
```

```
                hashMap.put("KEY_URL", "http://rcs.sichuanair.com");
                hashMap.put("KEY_BACKUP", "VALUE_ENABLE_BACKUP");
                hashMap.put("KEY_DELAY_MS_TIME", "2000");
                //这里调用DXRisk.getToken(DXRiskUtils.appId, hashMap)
                String token =
DXRisk.getToken("798feafe018020ed00e4b6f1d3fcf745", hashMap);
                TokenCallback tokenCallback2 = TokenCallback.this;
                if (tokenCallback2 != null) {
                    tokenCallback2.onToken(token);
                }
            }
        }.start();
    }


//DXRisk.getToken("798feafe018020ed00e4b6f1d3fcf745", hashMap)
 public static String getToken(String str, HashMap<String, String> hashMap) {
        try {
            //这里调用DXRiskInternal.getToken(str, hashMap);
            return DXRiskInternal.getToken(str, hashMap);
        } catch (LinkageError e) {
            c.a(a.a(), e);
            throw null;
        }
    }


//DXRiskInternal.getToken(str, hashMap);
public static String getToken(String str, HashMap<String, String> hashMap) {
        if (l.c(str)) {
            //这里经过hook发现是返回的false，所以没进来，直接return了
            str = mAppId;
        }
        if (hashMap == null) {
            //这里hashMap不为空
            hashMap = mParamsMap;
        }
    //这里继续调用 getTokenInternal(str, hashMap, true)生成token
        return getTokenInternal(str, hashMap, true);
    }


// getTokenInternal(str, hashMap, true);
public static String getTokenInternal(String str, HashMap<String, String>
hashMap, boolean z) {
        //调用x.l去生成token
        return (String) x.l(2, str, hashMap, Boolean.valueOf(z));
    }

//x.l
public static native Object l(int i, Object... objArr);
这个好像是项象的，目前(2025.3.9)测试都是固定的，明天重启一下，再测试测试
    2025.3.10又进行测试，针对于这个设备是固定的，那就先暂且认为是固定的
```

所以请求体中的riskReqBody是固定的，跟设备有关

所以这个请求体的明文就搞定了

登录接口总结

```
请求网址：
https://mapi.sichuanair.com/zt/tribeport/clientapi/login/member/codeLogin.htm
请求方式：POST
请求头：
    Cookie              JSESSIONID=838DF1059ECB8D42CA751AA6A561049F
    Content-Length      1100
    Content-Type        text/plain; charset=ISO-8859-1
    Host                mapi.sichuanair.com
    Connection          Keep-Alive
    User-Agent          Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
    Accept-Encoding     gzip
携带参数：
    actionType          clientapi/login/member/codeLogin
    platformID          0
    appVersion          6.13.1
    channelID           myself
    imei                5a9992681fc4b10a
    networkOperator
    mac                 02:00:00:00:00:00
    transActionId       517cf7c2-41f2-4705-a458-fe6048fbf9e9
请求体：
GKu4Leqtpn/v2wIlHP7J+WQStqM9H5DMRM11jRMwXSoDwsCOCKzI+Ou5DqukrBkcv4fUxk+hjZN55yS8
WqBpC986aG0sbcse7/r7oMIrvNJN+/I8OgIgTIzn1iRBlnN3L0NbV/1EGBX98tkbNCy6chSKPMNIMBr0
ulJH0S9xv5siLhNJH4G7+R3C/gA1lD+wGqnRE4I0HwAy/dZMVO4oS0PcVI41mRfGcDV3JihapHF6RiR5
eQnb+4sXaeKlbxJrZKQDHqA0nekjvdZZp0NBk4+95/2vmRKhP1/JjsX6XjCzJMDbiymZlJrSPboU1Agd
B5MtYmlXn0uhE9nfD8hMUychjPt6eMhcgba0iUC1gybbXjto1J+EmAncAleA9Yasp4IcKPIxA+zs83/N
/gT2x9hn5ZixJkdKYXqVuCTU+WBzG85sDQfsQg8LKtaR/1QeWFJfW3yvOguPz9ow+m1yDUEVWaLtIIPm
Jupjz8U91e9fKGHI5UUHektIZPGan9fanPj6rNBPs72lY/K+WGlyqA9JrxJKhGWf+fG93CuOxY5w6kuw
5rOlx67AADJGWyAOmXh26zTrLVSHt6OSBn+QMcN9ZXdAtltmO/2Qk+05c3T7lJZDYWepieRQFoHsnN4/
EYOnz7l3Qit46tnCXBp2LcLag1slcF3o3EV4/irKNL3ktOGifdJSz5lBVZVbGsG6Cb8II8uISbpvX6gB
vDSy/8nSH8ZQJw42f3IZMa/t6DplishOZEW97uo22W7ojjy7WiM154QAHcdSFMyLcRfoUN0nLpOpZ8y+
Tb7S3QqDbnbkaFVSm9XXXM6KxOg/3zOuQkF+ODdzwwpvJPRjeULI6sozSpEaAnYooCpcnCWgV3TyZeP4
+YbSbM6kM1R3jzfAcMZBuJvcbcOuiLI6EIXBUKa5M2GoWC9AEQ57pauDOoEH2DgcjFLaT7w/KcneDLVG
IMJ53IccfUn5aAEkUAfL+W2RslUAGJmFo1xbibsPqHbgIXtiHp+b6zpxTtX5xHj4jzpw9W5RvlIfrqg5
YFrJA5DYUs0AOqxAezjeeW1bJcuwOKVgc8TXkuh3vrJCbEiUhUKQl2JUhjM=
```

登录请求头

登录请求的请求头里有一个`Cookie`参数，这个参数我们前边分析过了，是给服务器发送请求，然后拿到的，他的作用就是给请求体，响应体加解密用的。它是desede加密的key。

```
请求头：
    Cookie              JSESSIONID=838DF1059ECB8D42CA751AA6A561049F
    Content-Length      1100
    Content-Type        text/plain; charset=ISO-8859-1
    Host                mapi.sichuanair.com
    Connection          Keep-Alive
    User-Agent          Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
    Accept-Encoding     gzip
```

关于`Cookie`参数请求的请求数据如下：

```
目标网址：https://mapi.sichuanair.com/zt/tribeport/encrypt_translate_key.htm
请求方式：POST
请求头：
    Content-Length      751
    Content-Type        text/plain; charset=ISO-8859-1
    Host                mapi.sichuanair.com
    Connection          Keep-Alive
    User-Agent          Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
    Accept-Encoding     gzip

请求体：
    {
    "body": {
        "encryptContent":
"kSXFqvJsCRM9rtia/DNVZW77NsZGhDWaOdJrVkv3xtTSpqT8QzqJYDhOjtx7CpPt/MblfOztzaPgYrd
StKwhgg==",
        "key":
"CoUnrcZI+s8glknCCWyuOE/fajOKIxCW2VQFe4o8/g7+ictmzwtmDiRWe4ZDDsLMQzUv7YSoj5KcXgl
foIemf8uyddFimYcDnh7XBwqIDIqv+wP/AliiJTmtYMhMO/BVg/6zFEArQgpXkw5vanjgxDPDjjThtHs
GWQqyA+slbDU="
    },
    "head": {
        "action": "ENCRYPT_TRANSLATE_KEY",
        "channelId": "myself",
        "platformId": 0,
        "proVersion": "2.0",
        "sign":
"2e93b528dea6d2c517a993d645c2ee539754c8515aa2c049b4ef143893d2053b929b62a938be71c
de57a3d2eeb90074d08c202c943a3789fab5f34cde1fd723cad3cb0ac43183b32fb464bc6bd37c5e
f63d9bfc4effd08c29e561f5322b7fdd9fcc06a4399bb36ffe9f4132cd7319e4adf5c6514d304e84
e659c377458252558",
        "timestamp": "2025-03-11 18:32:24",
        "transActionId": "2f44daf9-3a3b-4d2e-99b9-473d901bd4ef"
    }
}
携带参数：
    actionType  ENCRYPT_TRANSLATE_KEY
    platformID  0
```

```
    appVersion  6.13.1
    channelID   myself
    imei    5a9992681fc4b10a
    networkOperator
    mac 02:00:00:00:00:00
    transActionId   2f44daf9-3a3b-4d2e-99b9-473d901bd4ef
```

拿Cookie的请求头

```
Content-Length      751
    Content-Type        text/plain; charset=ISO-8859-1
    Host                mapi.sichuanair.com
    Connection          Keep-Alive
    User-Agent          Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
    Accept-Encoding     gzip
```

不需要动，固定就行

拿Cookie的请求体


```
{
    "body": {
        "encryptContent":
"kSXFqvJsCRM9rtia/DNVZW77NsZGhDWaOdJrVkv3xtTSpqT8QzqJYDh0jtx7CpPt/MblfOztzaPgYrd
StKwhgg==",
        "key":
"CoUnrcZI+s8glknCCWyuOE/fajOKIxCW2VQFe4o8/g7+ictmzwtmDiRWe4ZDDsLMQzUv7YSoj5KcXgl
foIemf8uyddFimYcDnh7XBwqIDIqv+wP/AliiJTmtYMhM0/BVg/6zFEArQgpXkw5vanjgxDPDjjThtHs
GWQqyA+slbDU="
    },
    "head": {
        "action": "ENCRYPT_TRANSLATE_KEY",
        "channelId": "myself",
        "platformId": 0,
        "proVersion": "2.0",
        "sign":
"2e93b528dea6d2c517a993d645c2ee539754c8515aa2c049b4ef143893d2053b929b62a938be71c
de57a3d2eeb90074d08c202c943a3789fab5f34cde1fd723cad3cb0ac43183b32fb464bc6bd37c5e
f63d9bfc4effd08c29e561f5322b7fdd9fcc06a4399bb36ffe9f4132cd7319e4adf5c6514d304e84
e659c377458252558",
        "timestamp": "2025-03-11 18:32:24",
        "transActionId": "2f44daf9-3a3b-4d2e-99b9-473d901bd4ef"
    }
}
```


首先body部分：

```
info：&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681
            body中的key：通过rsa对info进行加密后转为base64编码
            body中的encryptContent：desede/CBC/PKCS5Padding加密后转为base64编码，
                其中key为：&platformID=0&appVersion，
                iv为：01234567
```

head部分：

```
    actionType：ENCRYPT_TRANSLATE_KEY
    platformID：0
    appVersion：6.13.1
    channelID：myself
    imei固定的(跟设备信息有关):5a9992681fc4b10a
    networkOperator(固定的)：空
    mac(固定的)：02:00:00:00:00:00
    transActionId：uuid
    timestamp:时间戳
```

拿Cookie的请求参数

```
actionType  ENCRYPT_TRANSLATE_KEY
platformID  0
appVersion  6.13.1
channelID   myself
imei        5a9992681fc4b10a
networkOperator
mac         02:00:00:00:00:00
transActionId   UUID
```

拿Cookie的请求代码

```
import base64
import hashlib
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
import time
import uuid
from Crypto.Cipher import DES3
from Crypto.Util.Padding import pad
import requests


def desede_encrypt(plaintext, key):
    # 检查密钥长度是否符合要求，DESede 密钥长度必须是 16 或 24 字节
    if len(key) not in [16, 24]:
        raise ValueError("密钥长度必须为 16 或 24 字节")
    # 将写死的 IV 定义为字节类型
    iv = b'01234567'
    # 如果传入的明文是字符串类型，将其编码为字节类型
```

```python
    if isinstance(plaintext, str):
        plaintext = plaintext.encode('utf-8')
    # 创建 DES3 加密器对象，采用 CBC 模式和预先定义的 IV
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    # 对明文进行 PKCS5 填充，使其长度为块大小的整数倍
    padded_plaintext = pad(plaintext, DES3.block_size)
    # 执行加密操作
    encrypted_data = cipher.encrypt(padded_plaintext)
    return encrypted_data
def base64_encode(input_data):
    # 判断输入的数据类型
    if isinstance(input_data, str):
        # 如果是字符串，将其编码为字节类型
        input_bytes = input_data.encode('utf-8')
    elif isinstance(input_data, bytes):
        # 如果是字节类型，直接使用
        input_bytes = input_data
    else:
        # 如果输入既不是字符串也不是字节类型，抛出异常
        raise ValueError("Input must be either a string or bytes.")

    # 使用 base64 库进行编码
    encoded_bytes = base64.b64encode(input_bytes)
    # 将编码后的字节类型转换为字符串类型
    encoded_string = encoded_bytes.decode('utf-8')
    return encoded_string
def bytes_to_hex(bytes_data):
    return bytes_data.hex()
def rsa_encrypt(plain_text, public_key_str):
    # 解码Base64公钥
    public_key_bytes = base64.b64decode(public_key_str)
    # 加载DER格式的公钥
    public_key = serialization.load_der_public_key(
        public_key_bytes,
        backend=default_backend()
    )

    # 获取RSA公钥参数
    n = public_key.public_numbers().n
    e = public_key.public_numbers().e

    # 计算模长的字节数
    modulus_bytes = (n.bit_length() + 7) // 8

    # 将明文转换为字节并进行填充
    plain_bytes = plain_text.encode('utf-8')

    # 检查长度是否超过模长
    if len(plain_bytes) > modulus_bytes:
        raise ValueError(f"明文过长 ({len(plain_bytes)} > {modulus_bytes} bytes)")

    # 使用零字节在前方填充至模长
    padded = b'\x00' * (modulus_bytes - len(plain_bytes)) + plain_bytes

    # 将填充后的字节转换为整数
```

```
    m_int = int.from_bytes(padded, byteorder='big')

    # 执行RSA加密：c = m^e mod n
    c_int = pow(m_int, e, n)

    # 将加密结果转换回字节
    encrypted_bytes = c_int.to_bytes(modulus_bytes, byteorder='big')

    return encrypted_bytes
def getSign(inputStr):
    md5 = hashlib.md5()
    md5.update(inputStr.encode('utf-8'))
    md5_hex_str = md5.hexdigest()
    public_key_str =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+Qkd
MYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg
7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB"
    encrypted_bytes = rsa_encrypt(md5_hex_str, public_key_str)
    sign_hex = bytes_to_hex(encrypted_bytes)
    return  sign_hex
def getTimestamp():
    # 获取当前时间戳
    timestamp = time.time()  # 示例输出：1742002245.123456
    # 转换为本地时间并格式化
    local_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(timestamp))
    return local_time
def getTransActionId():
    # 生成一个UUID4（随机生成的UUID）
    generated_uuid = uuid.uuid4()
    return str(generated_uuid)
def getKey():
    inputStr = "&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681"
    public_key_str =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+Qkd
MYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg
7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB"
    rsa_result = rsa_encrypt(inputStr,public_key_str)
    return base64_encode(rsa_result)
def getEncryptContent():
    str = "&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681"
    key = "&platformID=0&appVersion"
    desede_result = desede_encrypt(str,key)
    return base64_encode(desede_result)
def connect_sign_plaintext(timestamp,transActionId):
    timestamp_cut = timestamp[:10]
    end_str = timestamp_cut+transActionId+"2822563731"
    return  end_str
def getSessionId():
    encryptContent = getEncryptContent()
    key = getKey()
    imei = "5a9992681fc4b10a"
    action = "ENCRYPT_TRANSLATE_KEY"
    networkOperator = ""
    mac = "02:00: 00:00: 00:00"
    channelId = "myself"
```

```
    platformId = "0"
    proVersion = "2.0"
    appVersion = "6.13.1"
    timestamp = getTimestamp()
    transActionId = getTransActionId()
    sign = getSign(connect_sign_plaintext(timestamp,transActionId))
    url = f"https://mapi.sichuanair.com/zt/tribeport/encrypt_translate_key.htm?
actionType={action}&platformID={platformId}&appVersion={appVersion}&channelID=
{channelId}&imei={imei}&networkOperator={networkOperator}&mac=
{mac}&transActionId={transActionId}"
    header = {
        "User-Agent":"Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1"
    }
    data = {
            "body": {
                "encryptContent": encryptContent,
                "key": key
            },
            "head": {
                "action": action,
                "channelId": channelId,
                "platformId": platformId,
                "proVersion":proVersion ,
                "sign": sign,
                "timestamp": timestamp,
                "transActionId": transActionId
            }
    }
    rep = requests.post(url,headers=header,json=data)
    header_cooie = rep.headers["Set-Cookie"][11:34]
    print(rep.headers["Set-Cookie"])
    print("header_cooie is =>",header_cooie)
    return header_cooie
def main():
    # timestamp = "2025-03-11"
    # trans_action_id = "2f44daf9-3a3b-4d2e-99b9-473d901bd4ef"
    # dex_crc = "2822563731"
    # str_concatenated = timestamp + trans_action_id + dex_crc
    # print("sign is =>",getSign(str_concatenated))
    # print(getTimestamp())
    # print(getTransActionId())
    # print(base64_encode("123456"))
    # print(connect_sign_plaintext(getTimestamp(),getTransActionId()))
    desede_key = getSessionId()



if __name__ == "__main__":
    main()
```

登录请求体

```
{"body":
{"mobileCountryCode":"86","phone":"15512103215","verifyCode":"123456"},"head":
{"action":"clientapi/login/member/codeLogin","appVersion":"6.13.1","channelId":"
myself","customerId":"","imie":"00000000-6fd2-5a75-ffff-
ffffca01fdf4","jpushId":"120c83f7611feca6238","macAddress":"02:00:00:00:00:00","
platformId":0,"proVersion":"2.0","sessionId":"0C9AE2425AD050460C7A910AE682BFEA",
"sign":"6f1ca25394a3e4b40b8c399d202efaf682f914a4cf73137f496d5f4ad5fab3ef9a3a5061
9641a801886f7424841e8abda86a2fc68edaa2fb4433d12454a18a2bafe8f4d8ac2e975f4c39181a
2d286009c1c028b238c44567f0ede4f218f15589d1dc740993a1e5a59054fb5901722ac8ddeeca62
ff90fc6552513b8df80cb82f","timestamp":"2025-03-09
16:43:13","transActionId":"0cbcb77b-6bf2-46ec-ae97-5cbb7b61fe37"},"riskReqBody":
{"constId":"67cd54a1h118yuhL38dC82jSn8w0GGc3EbwSZ3n3","validType":"0"}}
```


对这些字符串，进行3des加密后=>base64编码，然后就是请求体了

这里的明文又分为三部分：body、head、riskReqBody

请求体明文的body


```
"body":{"mobileCountryCode":"86","phone":"15512103215","verifyCode":"123456"}
```


就是国际区号、手机号、验证码

请求体明文的head


```
"head":
{"action":"clientapi/login/member/codeLogin","appVersion":"6.13.1","channelId":"
myself","customerId":"","imie":"00000000-6fd2-5a75-ffff-
ffffca01fdf4","jpushId":"120c83f7611feca6238","macAddress":"02:00:00:00:00:00","
platformId":0,"proVersion":"2.0","sessionId":"0C9AE2425AD050460C7A910AE682BFEA",
"sign":"6f1ca25394a3e4b40b8c399d202efaf682f914a4cf73137f496d5f4ad5fab3ef9a3a5061
9641a801886f7424841e8abda86a2fc68edaa2fb4433d12454a18a2bafe8f4d8ac2e975f4c39181a
2d286009c1c028b238c44567f0ede4f218f15589d1dc740993a1e5a59054fb5901722ac8ddeeca62
ff90fc6552513b8df80cb82f","timestamp":"2025-03-09
16:43:13","transActionId":"0cbcb77b-6bf2-46ec-ae97-5cbb7b61fe37"},
```


```
"action":"clientapi/login/member/codeLogin", 固定的
"appVersion":"6.13.1", 固定的
"channelId":"myself",固定的
"customerId":"",固定的
"imie":"00000000-6fd2-5a75-ffff-ffffca01fdf4", 由设备信息得到的uuid，这里可以固定
"jpushId":"120c83f7611feca6238", 固定的
"macAddress":"02:00:00:00:00:00",固定的
"platformId":0,固定的
"proVersion":"2.0",固定的
"sessionId":"0C9AE2425AD050460C7A910AE682BFEA", 通过getJSESSIONID去拿
"sign":"6f1ca25394a3e4b40b8c399d202efaf682f914a4cf73137f496d5f4ad5fab3ef9a3a5061
9641a801886f7424841e8abda86a2fc68edaa2fb4433d12454a18a2bafe8f4d8ac2e975f4c39181a
2d286009c1c028b238c44567f0ede4f218f15589d1dc740993a1e5a59054fb5901722ac8ddeeca62
ff90fc6552513b8df80cb82f", 通过getSign去拿
```

```
"timestamp":"2025-03-09 16:43:13", 通过getTimestamp去拿
"transActionId":"0cbcb77b-6bf2-46ec-ae97-5cbb7b61fe37" 通过getTransActionId去拿
```

请求体明文的riskReqBody

```
"riskReqBody":
{"constId":"67cd54a1h118yuhL38dC82jSn8w0GGc3EbwSZ3n3","validType":"0"}}
```

constId是固定的，validType是0

登录请求的代码

```python
import base64
import hashlib
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
import time
import uuid
from Crypto.Cipher import DES3
from Crypto.Util.Padding import pad
import requests
from Crypto.Util.Padding import unpad
import json

def str_to_json(json_str):
    """
    将字符串转换为 JSON 对象。

    :param json_str: 要转换的 JSON 字符串
    :return: 转换后的 JSON 对象（Python 字典或列表），如果转换失败返回 None
    """
    try:
        # 尝试将字符串转换为 JSON 对象
        json_obj = json.loads(json_str)
        return json_obj
    except json.JSONDecodeError:
        # 若字符串不是有效的 JSON 格式，捕获异常并打印错误信息
        print(f"输入的字符串 '{json_str}' 不是有效的 JSON 格式。")
        return None
def desede_decrypt(ciphertext, key):
    # 检查密钥长度是否符合要求，DESede 密钥长度必须是 16 或 24 字节
    if len(key) not in [16, 24]:
        raise ValueError("密钥长度必须为 16 或 24 字节")
    # 将写死的 IV 定义为字节类型，需与加密时的 IV 一致
    iv = b'01234567'
    # 创建 DES3 解密器对象，采用 CBC 模式和预先定义的 IV
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    # 执行解密操作
```

```python
    decrypted_data = cipher.decrypt(ciphertext)
    # 去除 PKCS5 填充
    unpadded_data = unpad(decrypted_data, DES3.block_size)
    try:
        # 尝试将解密后的字节数据解码为字符串
        result = unpadded_data.decode('utf-8')
        return result
    except UnicodeDecodeError:
        # 如果解码失败，返回字节数据
        return unpadded_data
def desede_encrypt(plaintext, key):
    # 检查密钥长度是否符合要求，DESede 密钥长度必须是 16 或 24 字节
    if len(key) not in [16, 24]:
        raise ValueError("密钥长度必须为 16 或 24 字节")
    # 将写死的 IV 定义为字节类型
    iv = b'01234567'
    # 如果传入的明文是字符串类型，将其编码为字节类型
    if isinstance(plaintext, str):
        plaintext = plaintext.encode('utf-8')
    # 创建 DES3 加密器对象，采用 CBC 模式和预先定义的 IV
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    # 对明文进行 PKCS5 填充，使其长度为块大小的整数倍
    padded_plaintext = pad(plaintext, DES3.block_size)
    # 执行加密操作
    encrypted_data = cipher.encrypt(padded_plaintext)
    return encrypted_data
def base64_encode(input_data):
    # 判断输入的数据类型
    if isinstance(input_data, str):
        # 如果是字符串，将其编码为字节类型
        input_bytes = input_data.encode('utf-8')
    elif isinstance(input_data, bytes):
        # 如果是字节类型，直接使用
        input_bytes = input_data
    else:
        # 如果输入既不是字符串也不是字节类型，抛出异常
        raise ValueError("Input must be either a string or bytes.")

    # 使用 base64 库进行编码
    encoded_bytes = base64.b64encode(input_bytes)
    # 将编码后的字节类型转换为字符串类型
    encoded_string = encoded_bytes.decode('utf-8')
    return encoded_string
def base64_decode(encoded_data):
    # 判断输入的数据类型
    if isinstance(encoded_data, str):
        # 如果是字符串，将其编码为字节类型
        encoded_bytes = encoded_data.encode('utf-8')
    elif isinstance(encoded_data, bytes):
        # 如果是字节类型，直接使用
        encoded_bytes = encoded_data
    else:
        # 如果输入既不是字符串也不是字节类型，抛出异常
        raise ValueError("Input must be either a string or bytes.")
```

```python
    try:
        # 使用 base64 库进行解码
        decoded_bytes = base64.b64decode(encoded_bytes)
        try:
            # 尝试将解码后的字节数据解码为字符串
            decoded_string = decoded_bytes.decode('utf-8')
            return decoded_string
        except UnicodeDecodeError:
            # 如果解码失败，返回字节数据
            return decoded_bytes
    except base64.binascii.Error:
        # 如果 Base64 解码失败，抛出异常
        raise ValueError("Invalid Base64-encoded input.")
def bytes_to_hex(bytes_data):
    return bytes_data.hex()
def rsa_encrypt(plain_text, public_key_str):
    # 解码Base64公钥
    public_key_bytes = base64.b64decode(public_key_str)
    # 加载DER格式的公钥
    public_key = serialization.load_der_public_key(
        public_key_bytes,
        backend=default_backend()
    )

    # 获取RSA公钥参数
    n = public_key.public_numbers().n
    e = public_key.public_numbers().e

    # 计算模长的字节数
    modulus_bytes = (n.bit_length() + 7) // 8

    # 将明文转换为字节并进行填充
    plain_bytes = plain_text.encode('utf-8')

    # 检查长度是否超过模长
    if len(plain_bytes) > modulus_bytes:
        raise ValueError(f"明文过长 ({len(plain_bytes)} > {modulus_bytes} bytes)")

    # 使用零字节在前方填充至模长
    padded = b'\x00' * (modulus_bytes - len(plain_bytes)) + plain_bytes

    # 将填充后的字节转换为整数
    m_int = int.from_bytes(padded, byteorder='big')

    # 执行RSA加密：c = m^e mod n
    c_int = pow(m_int, e, n)

    # 将加密结果转换回字节
    encrypted_bytes = c_int.to_bytes(modulus_bytes, byteorder='big')

    return encrypted_bytes
def getSign(inputStr):
    md5 = hashlib.md5()
    md5.update(inputStr.encode('utf-8'))
    md5_hex_str = md5.hexdigest()
```

```
    public_key_str =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+Qkd
MYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg
7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB"
    encrypted_bytes = rsa_encrypt(md5_hex_str, public_key_str)
    sign_hex = bytes_to_hex(encrypted_bytes)
    return  sign_hex
def getTimestamp():
    # 获取当前时间戳
    timestamp = time.time()  # 示例输出：1742002245.123456
    # 转换为本地时间并格式化
    local_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(timestamp))
    return local_time
def getTransActionId():
    # 生成一个UUID4（随机生成的UUID）
    generated_uuid = uuid.uuid4()
    return str(generated_uuid)
def getKey():
    inputStr = "&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681"
    public_key_str =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+Qkd
MYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg
7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB"
    rsa_result = rsa_encrypt(inputStr,public_key_str)
    return base64_encode(rsa_result)
def getEncryptContent():
    str = "&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681"
    key = "&platformID=0&appVersion"
    desede_result = desede_encrypt(str,key)
    return base64_encode(desede_result)
def connect_sign_plaintext(timestamp,transActionId):
    timestamp_cut = timestamp[:10]
    end_str = timestamp_cut+transActionId+"2822563731"
    return  end_str
def getJSESSIONID():
    encryptContent = getEncryptContent()
    key = getKey()
    imei = "5a9992681fc4b10a"
    action = "ENCRYPT_TRANSLATE_KEY"
    networkOperator = ""
    mac = "02:00: 00:00: 00:00"
    channelId = "myself"
    platformId = "0"
    proVersion = "2.0"
    appVersion = "6.13.1"
    timestamp = getTimestamp()
    transActionId = getTransActionId()
    sign = getSign(connect_sign_plaintext(timestamp,transActionId))
    url = f"https://mapi.sichuanair.com/zt/tribeport/encrypt_translate_key.htm?
actionType={action}&platformID={platformId}&appVersion={appVersion}&channelID=
{channelId}&imei={imei}&networkOperator={networkOperator}&mac=
{mac}&transActionId={transActionId}"
    header = {
```

```
        "User-Agent":"Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1"
    }
    data = {
            "body": {
                "encryptContent": encryptContent,
                "key": key
            },
            "head": {
                "action": action,
                "channelId": channelId,
                "platformId": platformId,
                "proVersion":proVersion ,
                "sign": sign,
                "timestamp": timestamp,
                "transActionId": transActionId
            }
    }
    rsp = requests.post(url,headers=header,json=data)
    return rsp
def login_code():
    mobileCountryCode = "86"
    phone = input("请输入你的手机号:")
    verifyCode = input("请输入你的验证码:")
    constId = "67cd54a1h118yuhL38dC82jSn8w0GGc3EbwSZ3n3"
    validType = "0"
    action = "clientapi/login/member/codeLogin"
    appVersion = "6.13.1"
    channelId = "myself"
    customerId = ""
    imie = "00000000-6fd2-5a75-ffff-ffffca01fdf4"
    jpushId = "120c83f7611feca6238"
    macAddress = "02:00:00:00:00:00"
    platformId = "0"
    proVersion = "2.0"
    networkOperator = ""
    sessionId = getJSESSIONID().headers["Set-Cookie"][11:43]
    timestamp = getTimestamp()
    transActionId = getTransActionId()
    sign = getSign(connect_sign_plaintext(timestamp,transActionId))
    url =
f"https://mapi.sichuanair.com/zt/tribeport/clientapi/login/member/codeLogin.htm?
actionType=clientapi/login/member/codeLogin&platformID={platformId}&appVersion=
{appVersion}&channelID={channelId}&imei={imie}&networkOperator=
{networkOperator}&mac={macAddress}&transActionId={transActionId}"
    body = f'"body":{{"mobileCountryCode":"{mobileCountryCode}","phone":"
{phone}","verifyCode":"{verifyCode}"}},'
    header = f'"head":{{"action":"{action}","appVersion":"
{appVersion}","channelId":"{channelId}","customerId":"{customerId}","imie":"
{imie}","jpushId":"{jpushId}","macAddress":"{macAddress}","platformId":
{platformId},"proVersion":"{proVersion}","sessionId":"{sessionId}","sign":"
{sign}","timestamp":"{timestamp}","transActionId":"{transActionId}"}},'
    riskReqBody = f'"riskReqBody":{{"constId":"{constId}","validType":"
{validType}"}}'
```

```
    data = "{"+body+header+riskReqBody+"}"
    ciphertext = base64_encode(desede_encrypt(data,sessionId[:24]))
    encoded_data = ciphertext.encode('ISO-8859-1', errors='replace')
    header = {
    "Cookie": f"JSESSIONID={sessionId}",
    "Content-Length": "1100",
    "Content-Type": "text/plain; charset=ISO-8859-1",
    "Host": "mapi.sichuanair.com",
    "Connection": "Keep-Alive",
    "User-Agent": "Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel2
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1",
    "Accept-Encoding": "gzip"
    }
    rsp = requests.post(url,headers=header,data=encoded_data)
    return rsp


    # rsp = requests.post(url,headers=header,data=data)
def main():
    # 发送登录请求
    login_rsp = login_code()
    # 拿登录请求的desede的key
    login_des_key = login_rsp.request.headers["Cookie"][11:43][:24]
    # 拿到响应体，此时还是desede加密后又base64编码过的
    login_response_ciphertxt_base64=login_rsp.text
    # base64解码，desede解密
    login_response_plaintext=
desede_decrypt(base64_decode(login_response_ciphertxt_base64),login_des_key)
    # 转为json，方便拿吐丝信息
    login_response_plaintext_json = str_to_json(login_response_plaintext)
    # 打印吐丝信息
    print(login_response_plaintext_json)
    print(login_response_plaintext_json["body"]["message"]["value"])


if __name__ == "__main__":
    main()
```

发送验证码接口

一直没搞发送验证码的接口，刚刚测试的时候发了个请求，抓到了包：

```
请求网址：
https://mapi.sichuanair.com/zt/tribeport/clientapi/common/sendMsgCodeWithType.ht
m?
actionType=clientapi/common/sendMsgCodeWithType&platformID=0&appVersion=6.13.1&c
hannelID=myself&imei=5a9992681fc4b10a&networkOperator=&mac=02:00:00:00:00:00&tra
nsActionId=912b431f-1568-46f6-8863-887ab13a3f5f
请求方式：POST
请求头：
    Cookie              JSESSIONID=DC31CCE030FB605CFE65BE5DE55314BD
    Content-Length      1048
    Content-Type        text/plain; charset=ISO-8859-1
```

```
    Host                mapi.sichuanair.com
    Connection          Keep-Alive
    User-Agent          Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
    Accept-Encoding     gzip
请求体：加密前的
{
"body":{
"mobile":
{"plaintext":"15512103215"},"mobileCountryCode":"86","type":"CODE_LOGIN","verify
Code":"firstValid"
},

"head":
{"action":"clientapi/common/sendMsgCodeWithType","appVersion":"6.13.1","channelI
d":"myself","customerId":"","imie":"00000000-6fd2-5a75-ffff-
076ffffca01fdf4","jpushId":"120c83f7611feca6238","macAddress":"02:00:00:00:00:00
","platformId":0,"proVersion":"2.0","sessionId":"DC31CCE030FB605CFE65BE5DE55314B
D","sign":"25fa4c25823d3c95f031a0276d50f5074e7308ed543436c62629429d6e2228889b8cf
288ad29e020616a7b1fea849dc49144c77cb26205c4b646bf02a22e6315cf34c1c4a4d3305383796
d737ff84e77ca84e054f69adae589f9c4570197c2534cd138dcc19adfc2dc83d2f2956ae89731a7c
b8879f08fefe03379db6e144b92","timestamp":"2025-03-15
12:43:03","transActionId":"912b431f-1568-46f6-8863-887ab13a3f5f"}
}
请求参数：
    actionType      clientapi/common/sendMsgCodeWithType
    platformID      0
    appVersion      6.13.1
    channelID       myself
    imei            5a9992681fc4b10a
    networkOperator
    mac             02:00:00:00:00:00
    transActionId   912b431f-1568-46f6-8863-887ab13a3f5f
```

抓包的同时，我们hook了java层系统加密库，我们拿请求体，响应体搜一下：

```
请求体：
================================================================
Cipher.init.overload('int', 'java.security.Key', 'java.security.spec. ') is
called!
----------加密----------
 desede/CBC/PKCS5Padding init key  Utf8:  DC31CCE030FB605CFE65BE5D
----------------------------------------------------------------
 desede/CBC/PKCS5Padding init IV  Utf8:  01234567
================================================================
Cipher.doFinal.overload('[B') is called!
```

```
 desede/CBC/PKCS5Padding doFinal data  Utf8:  {"body":{"mobile":
{"plaintext":"15512103215"},"mobileCountryCode":"86","type":"CODE_LOGIN","verify
Code":"firstValid"},"head":
{"action":"clientapi/common/sendMsgCodeWithType","appVersion":"6.13.1","channelI
d":"myself","customerId":"","imie":"00000000-6fd2-5a75-ffff-
ffffca01fdf4","jpushId":"120c83f7611feca6238","macAddress":"02:00:00:00:00:00","
platformId":0,"proVersion":"2.0","sessionId":"DC31CCE030FB605CFE65BE5DE55314BD",
"sign":"25fa4c25823d3c95f031a0276d50f5074e7308ed543436c62629429d6e2228889b8cf288
ad29e020616a7b1fea849dc49144c77cb26205c4b646bf02a22e6315cf34c1c4a4d3305383796d73
7ff84e77ca84e054f69adae589f9c4570197c2534cd138dcc19adfc2dc83d2f2956ae89731a7cb88
79f08fefe03379db6e144b92","timestamp":"2025-03-15
12:43:03","transActionId":"912b431f-1568-46f6-8863-887ab13a3f5f"}}

------------------------------------------------------------
desede/CBC/PKCS5Padding doFinal result  Base64:
KiNW+2dJ/qh8sAUgWbQeCt8qwfHrNUkQXGkA8xNlL36Qr9i7nQE4FFky+dYMoPdvUTUvwkj7m5t1hWPu
TadvLYfzZ52m22/Clfksf6xnPPZU6EXXjo0QrNcM+oBYA55/yrXzDZnwgpd4zidClWCq79K1PnKeZZpG
M5K8ePe2w+mVr4HPMpUgQt78pDbEWZpiTv0So1PPiakSBOhiYkn92vZTTPl60/0R+qElmZzFIVQxOtTu
JBpG05STRgQV8yMyLCjbcmLA9KyCzdXBZk0IXDjkWFqyfYaL6gMT3WkYvuEsJAZ45XNJQZ2wjaBdGBxA
AGyJ2lDBOL3LIOmzKiHuq6Y5pMPirNySHLrKy3ugo2DTX9ybQTUO4DpJwYKK9WOgUa8k/lx5LlB0v6QU
6eotOHan7SLCHPmfuUq4E/qDzmWUMAc0CWS3/Z+sJCygJypnfB3wrVKaqeYliJynvj3/5SsDK1pnoHEa
Kess9AfY8TLn12R9vJnp9WAylhtRB0xwKuLRuMjZCMQ8YZ9s9j9lNljFENcyXRdjZfmB5xl+Bd0I8RTI
I5wuPov/M/k2KEKp15qVWS2Qy8NCM4OUXYDCGOKkJvN5SaCWNfS/q3GlY82wFSPYl5da6xCK8brhKJ6C
HmT6Q6NgyaCy7NAZnA5Q6StSc7e69nTREgeGgaB9mYCx9unvjKMq3GYRFAhGjvhb85pr069kXNwRw1xK
xsf3Qeb9uVkocE36xkdzAorF8NjuRpqnlLAr9OdtxA7gbIudz1R7gBjv1MS9X5vYEPzqk3B1QVp0AW4p
DjWc3CVNc8hxrKwm0TOO2EDpOrH44ftZM/FWS7qudmrV7fkxk8La9QVPezHtqLf1OpcnEDmZuvaLPmdz
/5Jz/xgmxgfOtVXwM5yg5RgSiLpoN5sug4gaUjhrEFg4T+/jGO/AaiJ609U6pG23K9jw8coLC1bn3Ppj
/SRtdStwwTmRRqMXhWJ6hhM93fRvMDYZZEScySDvW/miHa3EJF7iAkEmgWmZ+Ya9BQILuCorXLQMyvqv
esdpkA==
```

```
响应体:
Cipher.init.overload('int', 'java.security.Key', 'java.security.spec. ') is
called!
----------解密----------
 desede/CBC/PKCS5Padding init key  Utf8:  DC31CCE030FB605CFE65BE5D
------------------------------------------------------------
 desede/CBC/PKCS5Padding init IV  Utf8:  01234567
============================================================
Cipher.doFinal.overload('[B') is called!
 desede/CBC/PKCS5Padding doFinal data  Base64:
KiNW+2dJ/qi9Jgu9tI+OkkSsd/ebAxYz8FC7bgw6Z/jIJqayfxRmEtBRlj/o6WAiH3msrXzMV9/TjlIv
W986BqLHoW+685jJwWw9Z
Hncevyp42VdJ99rZqFO2VHXKlJueNLZQTtjXJJSDzfbg7yBrD+VRpYNbfETvKgtkCNNyX3ZGyxMl4+Dt
dZRFOldl3Ves7TsZ1Paoj7BVYZD7lg95/eQRmhJE1H12qHpmy/7TH4=
------------------------------------------------------------
 desede/CBC/PKCS5Padding doFinal result  Utf8: {"body":
{"hasMessage":false,"message":{"keyCode":0,"value":"成
功"},"status":"OK","timestamp":1742013797852},"head":
{"forceLogOut":false,"serverTime":"2025-03-15 12:43:17"}}
============================================================
```

大致看了看，发送验证码接口的参数和登录接口的基本上一样，我觉得直接去发请求应该也可以发。

请求头带着Cookie User-Agent就行；

请求参数之前都说过，然后拼接到请求网址后边就行

请求体的明文：body和head，不过在body里一些新增的参数。

请求头

```
    Cookie              JSESSIONID=DC31CCE030FB605CFE65BE5DE55314BD
    Content-Length      1048
    Content-Type        text/plain; charset=ISO-8859-1
    Host                mapi.sichuanair.com
    Connection          Keep-Alive
    User-Agent          Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1
    Accept-Encoding     gzip
```

都带着就行，cookie改一下即可

请求体的明文


```
{
    "body":{"mobile":
{"plaintext":"15512103215"},"mobileCountryCode":"86","type":"CODE_LOGIN","verify
Code":"firstValid"},

"head":
{"action":"clientapi/common/sendMsgCodeWithType","appVersion":"6.13.1","channelI
d":"myself","customerId":"","imie":"00000000-6fd2-5a75-ffff-
ffffca01fdf4","jpushId":"120c83f7611feca6238","macAddress":"02:00:00:00:00:00","
platformId":0,"proVersion":"2.0","sessionId":"DC31CCE030FB605CFE65BE5DE55314BD",
"sign":"25fa4c25823d3c95f031a0276d50f5074e7308ed543436c62629429d6e2228889b8cf288
ad29e020616a7b1fea849dc49144c77cb26205c4b646bf02a22e6315cf34c1c4a4d3305383796d73
7ff84e77ca84e054f69adae589f9c4570197c2534cd138dcc19adfc2dc83d2f2956ae89731a7cb88
79f08fefe03379db6e144b92","timestamp":"2025-03-15
12:43:03","transActionId":"912b431f-1568-46f6-8863-887ab13a3f5f"}
}
```


只有body需要添加两个参数：type和verifyCode

type对应的应该是类型，这里是验证码登录所以就是CODE_LOGIN，即登录的验证码

verifyCode对应的是第几次获取?应该是这样的

我们去模拟一下请求：

```python
def sendLoginCode(phone):
    mobileCountryCode = "86"
    action = "clientapi/common/sendMsgCodeWithType"
    appVersion = "6.13.1"
    channelId = "myself"
    customerId = ""
    imie = "00000000-6fd2-5a75-ffff-ffffca01fdf4"
    jpushId = "120c83f7611feca6238"
```

```
    macAddress = "02:00:00:00:00:00"
    platformId = "0"
    proVersion = "2.0"
    networkOperator = ""
    sessionId = getJSESSIONID().headers["Set-Cookie"][11:43]
    timestamp = getTimestamp()
    transActionId = getTransActionId()
    type= "CODE_LOGIN"
    verifyCode="firstValid"
    sign = getSign(connect_sign_plaintext(timestamp, transActionId))
    url =
f"https://mapi.sichuanair.com/zt/tribeport/clientapi/common/sendMsgCodeWithType.
htm"
    body = f'"body":{{"mobile":{{"plaintext":"{phone}"}},"mobileCountryCode":"
{mobileCountryCode}","type":"{type}","verifyCode":"{verifyCode}"}},'
    heade = f'"head":{{"action":"{action}","appVersion":"
{appVersion}","channelId":"{channelId}","customerId":"{customerId}","imie":"
{imie}","jpushId":"{jpushId}","macAddress":"{macAddress}","platformId":
{platformId},"proVersion":"{proVersion}","sessionId":"{sessionId}","sign":"
{sign}","timestamp":"{timestamp}","transActionId":"{transActionId}"}}'

    data = "{" + body + heade+ "}"
    ciphertext = base64_encode(desede_encrypt(data, sessionId[:24]))
    # encoded_data = str(ciphertext.encode('ISO-8859-1', errors='replace'))

    header = {
        "Cookie": f"JSESSIONID={sessionId}",
        "Content-Length": "1048",
        "Content-Type": "text/plain; charset=ISO-8859-1",
        "Host": "mapi.sichuanair.com",
        "Connection": "Keep-Alive",
        "User-Agent": "Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1",
        "Accept-Encoding": "gzip"
    }
    params = {
    "actionType": action,
    "platformID": platformId,
    "appVersion": appVersion,
    "channelID": channelId,
    "imie": imie,
    "networkOperator": networkOperator,
    "mac":macAddress,
    "transActionId": transActionId
}


    rsp = requests.post(url, headers=header, data=ciphertext,params=params)
    return  rsp
```

登录接口的模拟请求

完整代码

```
import base64
import hashlib
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
import time
import uuid
from Crypto.Cipher import DES3
from Crypto.Util.Padding import pad
import requests
from Crypto.Util.Padding import unpad
import json

def str_to_json(json_str):
    """
    将字符串转换为 JSON 对象。

    :param json_str: 要转换的 JSON 字符串
    :return: 转换后的 JSON 对象（Python 字典或列表），如果转换失败返回 None
    """
    try:
        # 尝试将字符串转换为 JSON 对象
        json_obj = json.loads(json_str)
        return json_obj
    except json.JSONDecodeError:
        # 若字符串不是有效的 JSON 格式，捕获异常并打印错误信息
        print(f"输入的字符串 '{json_str}' 不是有效的 JSON 格式。")
        return None
def desede_decrypt(ciphertext, key):
    # 检查密钥长度是否符合要求，DESede 密钥长度必须是 16 或 24 字节
    if len(key) not in [16, 24]:
        raise ValueError("密钥长度必须为 16 或 24 字节")
    # 将写死的 IV 定义为字节类型，需与加密时的 IV 一致
    iv = b'01234567'
    # 创建 DES3 解密器对象，采用 CBC 模式和预先定义的 IV
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    # 执行解密操作
    decrypted_data = cipher.decrypt(ciphertext)
    # 去除 PKCS5 填充
    unpadded_data = unpad(decrypted_data, DES3.block_size)
    try:
        # 尝试将解密后的字节数据解码为字符串
        result = unpadded_data.decode('utf-8')
        return result
    except UnicodeDecodeError:
        # 如果解码失败，返回字节数据
        return unpadded_data
def desede_encrypt(plaintext, key):
    # 检查密钥长度是否符合要求，DESede 密钥长度必须是 16 或 24 字节
    if len(key) not in [16, 24]:
        raise ValueError("密钥长度必须为 16 或 24 字节")
```

```python
    # 将写死的 IV 定义为字节类型
    iv = b'01234567'
    # 如果传入的明文是字符串类型，将其编码为字节类型
    if isinstance(plaintext, str):
        plaintext = plaintext.encode('utf-8')
    # 创建 DES3 加密器对象，采用 CBC 模式和预先定义的 IV
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    # 对明文进行 PKCS5 填充，使其长度为块大小的整数倍
    padded_plaintext = pad(plaintext, DES3.block_size)
    # 执行加密操作
    encrypted_data = cipher.encrypt(padded_plaintext)
    return encrypted_data
def base64_encode(input_data):
    # 判断输入的数据类型
    if isinstance(input_data, str):
        # 如果是字符串，将其编码为字节类型
        input_bytes = input_data.encode('utf-8')
    elif isinstance(input_data, bytes):
        # 如果是字节类型，直接使用
        input_bytes = input_data
    else:
        # 如果输入既不是字符串也不是字节类型，抛出异常
        raise ValueError("Input must be either a string or bytes.")

    # 使用 base64 库进行编码
    encoded_bytes = base64.b64encode(input_bytes)
    # 将编码后的字节类型转换为字符串类型
    encoded_string = encoded_bytes.decode('utf-8')
    return encoded_string
def base64_decode(encoded_data):
    # 判断输入的数据类型
    if isinstance(encoded_data, str):
        # 如果是字符串，将其编码为字节类型
        encoded_bytes = encoded_data.encode('utf-8')
    elif isinstance(encoded_data, bytes):
        # 如果是字节类型，直接使用
        encoded_bytes = encoded_data
    else:
        # 如果输入既不是字符串也不是字节类型，抛出异常
        raise ValueError("Input must be either a string or bytes.")

    try:
        # 使用 base64 库进行解码
        decoded_bytes = base64.b64decode(encoded_bytes)
        try:
            # 尝试将解码后的字节数据解码为字符串
            decoded_string = decoded_bytes.decode('utf-8')
            return decoded_string
        except UnicodeDecodeError:
            # 如果解码失败，返回字节数据
            return decoded_bytes
    except base64.binascii.Error:
        # 如果 Base64 解码失败，抛出异常
        raise ValueError("Invalid Base64-encoded input.")
def bytes_to_hex(bytes_data):
```

```
    return bytes_data.hex()
def rsa_encrypt(plain_text, public_key_str):
    # 解码Base64公钥
    public_key_bytes = base64.b64decode(public_key_str)
    # 加载DER格式的公钥
    public_key = serialization.load_der_public_key(
        public_key_bytes,
        backend=default_backend()
    )

    # 获取RSA公钥参数
    n = public_key.public_numbers().n
    e = public_key.public_numbers().e

    # 计算模长的字节数
    modulus_bytes = (n.bit_length() + 7) // 8

    # 将明文转换为字节并进行填充
    plain_bytes = plain_text.encode('utf-8')

    # 检查长度是否超过模长
    if len(plain_bytes) > modulus_bytes:
        raise ValueError(f"明文过长 ({len(plain_bytes)} > {modulus_bytes} bytes)")

    # 使用零字节在前方填充至模长
    padded = b'\x00' * (modulus_bytes - len(plain_bytes)) + plain_bytes

    # 将填充后的字节转换为整数
    m_int = int.from_bytes(padded, byteorder='big')

    # 执行RSA加密：c = m^e mod n
    c_int = pow(m_int, e, n)

    # 将加密结果转换回字节
    encrypted_bytes = c_int.to_bytes(modulus_bytes, byteorder='big')

    return encrypted_bytes
def getSign(inputStr):
    md5 = hashlib.md5()
    md5.update(inputStr.encode('utf-8'))
    md5_hex_str = md5.hexdigest()
    public_key_str =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+Qkd
MYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg
7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB"
    encrypted_bytes = rsa_encrypt(md5_hex_str, public_key_str)
    sign_hex = bytes_to_hex(encrypted_bytes)
    return  sign_hex
def getTimestamp():
    # 获取当前时间戳
    timestamp = time.time()  # 示例输出：1742002245.123456
    # 转换为本地时间并格式化
    local_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(timestamp))
    return local_time
def getTransActionId():
```

```
    # 生成一个UUID4（随机生成的UUID）
    generated_uuid = uuid.uuid4()
    return str(generated_uuid)
def getKey():
    inputStr = "&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681"
    public_key_str =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCdqp4yZcGX2yVCsM2itn3R35JW1rJwqEXHTHw+Qkd
MYKqFUo9svO7LD+U/tqXGjKeSu3oLc3B49P3j62Ex2w1As9Q75Ibf53fUkox4MwzwjaouMurpzwNwMJg
7BE+8zwAUJFZvwP7P/ses87N2nje/m/wy7Xm2zREkOfhfNAaY5QIDAQAB"
    rsa_result = rsa_encrypt(inputStr,public_key_str)
    return base64_encode(rsa_result)
def getEncryptContent():
    str = "&platformID=0&appVersion=6.13.1&channelID=myself&imei=5a9992681"
    key = "&platformID=0&appVersion"
    desede_result = desede_encrypt(str,key)
    return base64_encode(desede_result)
def connect_sign_plaintext(timestamp,transActionId):
    timestamp_cut = timestamp[:10]
    end_str = timestamp_cut+transActionId+"2822563731"
    return  end_str
def getJSESSIONID():
    encryptContent = getEncryptContent()
    key = getKey()
    imei = "5a9992681fc4b10a"
    action = "ENCRYPT_TRANSLATE_KEY"
    networkOperator = ""
    mac = "02:00: 00:00: 00:00"
    channelId = "myself"
    platformId = "0"
    proVersion = "2.0"
    appVersion = "6.13.1"
    timestamp = getTimestamp()
    transActionId = getTransActionId()
    sign = getSign(connect_sign_plaintext(timestamp,transActionId))
    url = f"https://mapi.sichuanair.com/zt/tribeport/encrypt_translate_key.htm?
actionType={action}&platformID={platformId}&appVersion={appVersion}&channelID=
{channelId}&imei={imei}&networkOperator={networkOperator}&mac=
{mac}&transActionId={transActionId}"
    header = {
        "User-Agent":"Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1"
    }
    data = {
            "body": {
                "encryptContent": encryptContent,
                "key": key
            },
            "head": {
                "action": action,
                "channelId": channelId,
                "platformId": platformId,
                "proVersion":proVersion ,
                "sign": sign,
                "timestamp": timestamp,
```

```
                "transActionId": transActionId
            }
    }
    rsp = requests.post(url,headers=header,json=data)
    return rsp
def sendLoginCode(phone):
    mobileCountryCode = "86"
    action = "clientapi/common/sendMsgCodeWithType"
    appVersion = "6.13.1"
    channelId = "myself"
    customerId = ""
    imie = "00000000-6fd2-5a75-ffff-ffffca01fdf4"
    jpushId = "120c83f7611feca6238"
    macAddress = "02:00:00:00:00:00"
    platformId = "0"
    proVersion = "2.0"
    networkOperator = ""
    sessionId = getJSESSIONID().headers["Set-Cookie"][11:43]
    timestamp = getTimestamp()
    transActionId = getTransActionId()
    type= "CODE_LOGIN"
    verifyCode="firstValid"
    sign = getSign(connect_sign_plaintext(timestamp, transActionId))
    url =
f"https://mapi.sichuanair.com/zt/tribeport/clientapi/common/sendMsgCodeWithType.
htm"
    body = f'"body":{{"mobile":{{"plaintext":"{phone}"}},"mobileCountryCode":"
{mobileCountryCode}","type":"{type}","verifyCode":"{verifyCode}"}},'
    heade = f'"head":{{"action":"{action}","appVersion":"
{appVersion}","channelId":"{channelId}","customerId":"{customerId}","imie":"
{imie}","jpushId":"{jpushId}","macAddress":"{macAddress}","platformId":
{platformId},"proVersion":"{proVersion}","sessionId":"{sessionId}","sign":"
{sign}","timestamp":"{timestamp}","transActionId":"{transActionId}"}}'

    data = "{" + body + heade+ "}"
    ciphertext = base64_encode(desede_encrypt(data, sessionId[:24]))
    # encoded_data = str(ciphertext.encode('ISO-8859-1', errors='replace'))

    header = {
        "Cookie": f"JSESSIONID={sessionId}",
        "Content-Length": "1048",
        "Content-Type": "text/plain; charset=ISO-8859-1",
        "Host": "mapi.sichuanair.com",
        "Connection": "Keep-Alive",
        "User-Agent": "Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1",
        "Accept-Encoding": "gzip"
    }
    params = {
    "actionType": action,
    "platformID": platformId,
    "appVersion": appVersion,
    "channelID": channelId,
    "imie": imie,
```

```
    "networkOperator": networkOperator,
    "mac":macAddress,
    "transActionId": transActionId
}


    rsp = requests.post(url, headers=header, data=ciphertext,params=params)
    return  rsp
def login_req(phone,verifyCode,sessionId):
    mobileCountryCode = "86"
    constId = "67d637c2yQnUEroozJZv8dbuWt2dEdV6iTOYgvZ3"
    validType = "0"
    action = "clientapi/login/member/codeLogin"
    appVersion = "6.13.1"
    channelId = "myself"
    customerId = ""
    imie = "00000000-6fd2-5a75-ffff-ffffca01fdf4"
    jpushId = "120c83f7611feca6238"
    macAddress = "02:00:00:00:00:00"
    platformId = "0"
    proVersion = "2.0"
    networkOperator = ""
    # sessionId = getJSESSIONID().headers["Set-Cookie"][11:43]
    timestamp = getTimestamp()
    transActionId = getTransActionId()
    sign = getSign(connect_sign_plaintext(timestamp,transActionId))
    url =
f"https://mapi.sichuanair.com/zt/tribeport/clientapi/login/member/codeLogin.htm"
    body = f'"body":{{"mobileCountryCode":"{mobileCountryCode}","phone":"
{phone}","verifyCode":"{verifyCode}"}},'
    head = f'"head":{{"action":"{action}","appVersion":"
{appVersion}","channelId":"{channelId}","customerId":"{customerId}","imie":"
{imie}","jpushId":"{jpushId}","macAddress":"{macAddress}","platformId":
{platformId},"proVersion":"{proVersion}","sessionId":"{sessionId}","sign":"
{sign}","timestamp":"{timestamp}","transActionId":"{transActionId}"}},'
    riskReqBody = f'"riskReqBody":{{"constId":"{constId}","validType":"
{validType}"}}'
    data = "{"+body+head+riskReqBody+"}"
    ciphertext = base64_encode(desede_encrypt(data,sessionId[:24]))
    # encoded_data = ciphertext.encode('ISO-8859-1', errors='replace')
    header = {
    "Cookie": f"JSESSIONID={sessionId}",
    "Content-Length": "1100",
    "Content-Type": "text/plain; charset=ISO-8859-1",
    "Host": "mapi.sichuanair.com",
    "Connection": "Keep-Alive",
    "User-Agent": "Mozilla/5.0 (Linux; U; Android 10; zh-cn; Pixel2
Build/QP1A.191005.007.A3) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0
Mobile Safari/533.1",
    "Accept-Encoding": "gzip"
    }

    params = {
        "actionType": action,
        "platformID": platformId,
```

```
        "appVersion": appVersion,
        "channelID": channelId,
        "imie": imie,
        "networkOperator": networkOperator,
        "mac": macAddress,
        "transActionId": transActionId
    }
    print("body plaintext is =>",data)
    rsp = requests.post(url,headers=header,data=ciphertext)
    return rsp


    # rsp = requests.post(url,headers=header,data=data)

def main():
    # 输入手机号
    phone = input("请输入你的手机号:")
    # 发送获取验证码请求
    sendLoginCodeRsp = sendLoginCode(phone)
    print("sendLoginCode request status:",sendLoginCodeRsp)
    # verifyCode = input("请输入您收到的验证码:")
    # print("sendcode cookie is =>",sendLoginCodeRsp.request.headers["Cookie"]
[11:43])
    # 发送登录请求
    # login_rsp =
login_req(phone,verifyCode,sendLoginCodeRsp.request.headers["Cookie"][11:43])
    # print("login request url is =>",login_rsp.request.url)
    # print("login request header is =>",login_rsp.request.headers)
    # print("login request body is =>",login_rsp.request.body)
    # 拿登录请求的desede的key
    # login_des_key = login_rsp.request.headers["Cookie"][11:43][:24]
    # 拿到响应体，此时还是desede加密后又base64编码过的
    # login_response_ciphertxt_base64=login_rsp.text
    # base64解码，desede解密
    # login_response_plaintext=
desede_decrypt(base64_decode(login_response_ciphertxt_base64),login_des_key)
    # print(login_response_plaintext)




if __name__ == "__main__":
    main()
```

各函数介绍

```
def str_to_json(json_str)
# 字符串转为json
def desede_decrypt(ciphertext, key)
# desede加密，输入明文和key
def desede_encrypt(plaintext, key)
# desede解密，输入密文和key
def base64_encode(input_data)
# base64编码函数
```

```
def base64_decode(encoded_data)
# base64解码函数
def bytes_to_hex(bytes_data)
# byte转hex
def rsa_encrypt(plain_text, public_key_str)
# rsa加密，输入明文和公钥
def getSign(inputStr)
# 获取sign，输入拼接好的字符串
def getTimestamp()
# 获取固定格式的时间戳
def getTransActionId()
# 获取TransActionID，其实就是uuid
def getKey()
# 获取发送拿seesionid的请求体的head的参数key
def getEncryptContent()
# 获取发送拿seesionid的请求体的head的参数EncryptContent
def connect_sign_plaintext(timestamp,transActionId)
# 进行拼接生成sign的字符串，
def getJSESSIONID()
# 发送请求，拿到sessionid
def sendLoginCode(phone)
# 发送请求，发送手机验证码
def login_req(phone,verifyCode,sessionId)
# 发送请求，进行验证码登录
def main()
# 调用函数，完成 输入手机号、发送验证码、输入验证码、登录操作
```