全国信用信息共享平台公用事业
信息归集接口说明

二〇二零年八月

一、概述

根据《公共信用信息资源目录》要求，希望各地按照信用信息归集目录提供数据，提供方式有两种，一是在前置机发布库表资源，由国家信用中心订阅；二是提供接口的方式，由国家信用中心调用。其中接口的方式主要针对敏感信息和数据量大的信息。目前法人中的纳税信息、欠税信息、社保缴费信息、公积金缴存信息、公用事业缴费信息、拖欠公用事业费用信息，自然人中的纳税信息、欠税信息、社保缴费信息、公积金个人缴存信息、公积金个人贷款信息、公用事业缴费信息、拖欠公用事业费用信息等要求采用接口方式提供。各地完成接口开发后，填写接口登记表，并发给国家信用信息中心。

二、接口通用要求

1. 通用接口要求

1.1. 接口调用流程描述

1、 相关接口采用 RSA 加密 / 解密，生成公钥 / 密钥对（参见“生成公钥/密钥”代码 CreateSecretKey.getPublicKeyStr 方法和 CreateSecretKey.getPrivateKeyStr 方法）。

2、 生成用户签名，用户签名为请求方用户名+密码+时间戳 Json 字符串用 RSA 加密算法使用己方密钥加密获得（参见“RSA 加密解密算法”代码 RSAUtil.encrypt 方法）。

3、 准备请求头（header），包含模块 id、用户签名。

4、 准备请求内容（body），将请求参数对象转成 JSON 格式，将 JSON 使用 RSA 加密算法使用己方密钥加密（参见“RSA 加密解密算法”代码 RSAUtil.encrypt 方法）。

5、 调用方发起请求（参见“通用接口请求/响应示例”代码）。

6、 获得接口返回（参见“通用接口请求/响应示例”代码）。

7、 将业务节点 body 部分进行解密（参见“RSA 加密解密算法”代码 RSAUtil.decrypt 方法）。

1.2. 接口信息描述

接口地址（根据 APPID 指定）	http://IP:Port/Path
请求方法	Post
请求内容类型	application/json; charset=utf-8
请求格式	Json 字符串
返回格式	Json 字符串

1.3. 接口请求参数

接口请求包含两部分：请求头和请求参数，接口模块系统的参数和签名认证参数通过请求头传递；接口请求的业务参数通过请求参数传递，业务参数需要进行加密。

请求头（Header）：

序号	参数	参数名称	是否为空	类型	长度	说明
1	appsign	请求签名	否	字符串		根据请求方用户名+密码+时间戳 Json 字符串用 RSA 加密得到。加密方法参见文中描述。

请求参数（根据不同接口模块不同）：

序号	参数	参数名称	是否为空	类型	长度	说明
1	reqparams	请求字符串		字符串		使用请求方密钥进行 RSA 加密后的 json 格式的请求字符串。

加密前的请求数据格式：

{ "qymc":"ABC 公司","tyshxydm":"1234567890123456 78"}

加密后的请求数据格式：

{ "reqparams":"jmjBivZg/9JwXasTkv6OhJdlVShhl3iX3TFkrkKmLvNx2R7fAij18Pitf81EVI+stwtYFY5cOaFg2sag1NzPJK5vGVgB36fW0q4o5RJQVPvOHsrk058Gu3uA3N36wPS+wTYCUCFkvhnpE5SKJ2kZsqVbl68a7sgBmGzTnJ2w0w0="}

1.4. 接口返回参数

接口返回结果以 json 格式的 RSA 加密字符串信息（String）返回。响应返回值：

序号	参数	参数名称	是否为空	类型	长度	说明

序号	参数	参数名称	是否为空	类型	长度	说明
1	resinfo	响应返回值集合		字符串		RSA 加密的 Json 格式的字符串返回值。

返回信息的格式：

{
"resinfo":"jmjBivZg/9JwXasTkv6OhJdlVShhl3iX3TFkrkKmLvNx2R7fAij18Pitf81EVI+stwtYFY5cOaFg2sag1NzPJK5vGVgB36fW0q4o5RJQVPvOHsrk058Gu3uA3N36wPS+wTYCUCFkvhnpE5SKJ2kZsqVbl68a7sgBmGzTnJ2w0w0="
}

将返回值用发送方公共密钥进行 RSA 解密后得到真正的响应返回值域。解密后的响应返回值域：

序号	参数	参数名称	是否为空	类型	长度	说明
1	state	响应返回状态码		字符串		
2	timestamp	响应返回时间戳				
3	result	返回信息	否	字符串		成功提示信息、错误提示信息
4	datas	响应返回值集合		字符串		Json 格式的 1 条至多条返回值记录集（不超过 10 条记录）。

解密后请求失败的返回数据格式：

{
"state":"e-109","timestamp":"1234567890","result":"请求超时"
}

解密后请求成功的返回数据格式：

{
"state":"s-1","timestamp":"1234567890","result":"调用成功",
"datas":[{"qymc":"ABC 公 司 ","tyshxydm":"123456780912345678","phone":"010-12345678"},{" qymc":" ABC 公 司 ","tyshxydm":"123456780912345678","phone":"010-12345666"}]
}

1.5. 通用接口请求/响应示例

通过 RSA 加密解密算法生成，接口提供方与接口调用方都使用此类生成公钥/密钥对，

并向对方公布公钥，在请求时使用己方密钥进行加密，获得返回结果时使用对端公钥进行解密。

```
import com.alibaba.fastjson.JSONObject;
import okhttp3.*;
import cn.les.utils.RSAUtil;

/**
 * 信用信息共享平台接口 Demo
 */
public class ClientDemo {
    static String api_url = "http://IP:Port/api";
    public static String  YU_PUBLIC_KEY_STRING  = ""; //对端公钥
    private static String MY_PRIVATE_KEY_STRING = ""; //己方私钥
    private String username = "";
    private String password = "";

    private void initKey() throws Exception {
        //todo 从文件中获得有效的对端公钥、己方密钥
        MY_PRIVATE_KEY_STRING = "MIIC......ZdDA==";
        YU_PUBLIC_KEY_STRING = "MIGf......AQAB ";
        //todo 从数据库中获得用户信息
        username = "user01";
        password = "123456";
    }

    /**
     * 生成签名
     */
    private String getAppsign() throws Exception {
        JSONObject data=new JSONObject();
        data.put("username",username);
        data.put("password",password);
        data.put("timespamp",""+System.currentTimeMillis());
        String strEnc = data.toJSONString();
        strEnc = RSAUtil.encrypt(MY_PRIVATE_KEY_STRING,new
String(strEnc.getBytes("UTF-8"))); //进行 RSA 加密
        return strEnc;
    }

    /**
     * 生成参数
     */
    private String genParams(JSONObject data) throws Exception {
```

```
        String strEnc = data.toJSONString();
        System.out.println("参数： "+strEnc);
        strEnc = RSAUtil.encrypt(MY_PRIVATE_KEY_STRING,new
String(strEnc.getBytes("UTF-8"))); //进行 RSA 加密
        System.out.println("RSA 参数： "+strEnc);
        JSONObject requestParams = new JSONObject();
        requestParams.put("reqparams", strEnc);
        return requestParams.toJSONString();
    }

    /**
     * 解析返回值
     */
    private String parseRes(Response response) throws Exception {
        JSONObject objRes = JSONObject.parseObject(response.body().string());
        String strDec = objRes.get("body").toString();
        System.out.println("返回 body: "+strDec);
        strDec = RSAUtil.decrypt(YU_PUBLIC_KEY_STRING, strDec);
        return strDec;
    }

    public static void main(String[] args) throws Exception {
        ClientDemo obj = new ClientDemo();
        obj.initKey();
        JSONObject paramData=new JSONObject();
        String productID="frgysyjf";
        paramData.put("tyshxydm","91110000710929498G");
        String appSign = obj.getAppsign();

        OkHttpClient client = new OkHttpClient();
        MediaType mediaType = MediaType.parse("application/json; charset=UTF-8");
        RequestBody body = RequestBody.create(mediaType, obj.genParams(paramData));
        Request request = new Request.Builder()
                .url(api_url + "/" + productID) //假设 restful 服务发布地址是:
http://IP:Port/api/frgysyjf
                .addHeader("Content-Type", "application/json charset=UTF-8")
                .addHeader("appid", productID)
                .addHeader("appsign", appSign)
                .post(body)
                .build();
        Response response = client.newCall(request).execute();
        System.out.println(obj.parseRes(response));
    }
}
```

2. 安全要求

信用信息共享平台和接口提供方通过 RSA 非对称加密方式加密签名及请求参数，返回结果记录集也通过 RSA 非对称加密方式加密。

通过解密签名获得签名的用户名/密码和时间戳信息，用户名/密码用于进行身份认证和访问权限检查，时间戳可用来控制请求时间范围。

信用信息共享平台和接口提供方双方约定密钥和公钥的有效期限，定期更新各自的公钥/密钥对并维护各自的公钥信息，应使用共享平台提供的方法生成公钥/密钥对（参见方法）。

2.1. RSA 加密解密算法

```
import org.apache.commons.codec.binary.Base64;
import javax.crypto.Cipher;
import java.io.ByteArrayOutputStream;
import java.security.*;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.RSAPrivateKeySpec;
import java.security.spec.RSAPublicKeySpec;
import java.security.spec.X509EncodedKeySpec;

/*
 * RSA 加密和解密
 */
public class RSAUtil {

    private static final int MAX_ENCRYPT_BLOCK = 117;
    private static final int MAX_DECRYPT_BLOCK = 128;

    private static String MY_PRIVATE_KEY_STRING = "";
    private static String MY_PUBLIC_KEY_STRING = "";

    public static void initKey() {
        //todo 从文件中获得有效的公钥/密钥对
        MY_PRIVATE_KEY_STRING = "MII......jyc="; //我方密钥
        MY_PUBLIC_KEY_STRING = "MIG......QAB"; //我方公钥
    }
```

```
    /**
     * 使用对端公钥对加密字符串进行 RSA 解码(缺省 Win 平台)
     *
     * @param publicKeyText 对端公钥
     * @param cipherText    待解密字符串
     * @return 缺省解密值
     */
    public static String decrypt(String publicKeyText, String cipherText) throws Exception
{
        return decrypt(publicKeyText, cipherText, true);
    }

    /**
     * 使用对端公钥对加密字符串进行 RSA 解码(指定平台)
     *
     * @param publicKeyText 对端公钥
     * @param cipherText    待解密字符串
     *@param isWin          是否 Windows 平台，Linux 需要对解码字符串进行 Base64 计算（加密使
用 Base64 时解码也需要使用）
     * @return 指定平台解密值
     */
    public static String decrypt(String publicKeyText, String cipherText, boolean isWin)
throws Exception {
        PublicKey publicKey = getPublicKey(publicKeyText);
        if (isWin) return decrypt(publicKey, cipherText);
        else return decryptByBase64(publicKey, cipherText); //Linux 需要对解码字符串进行
Base64 计
    }

    /**
     * 使用己端私钥对字符串进行 RSA 加密(缺省 Win 平台)
     *
     * @param privateKey 己端私钥
     * @param plainText  待加密字符串
     * @return 缺省加密值
     */
    public static String encrypt(String privateKey, String plainText) throws Exception {
        return encrypt(privateKey, plainText, true);
    }

    /**
     * 使用己端私钥对字符串进行 RSA 加密(指定平台)
     *
```

```
     * @param privateKey 已端私钥
     * @param plainText  待加密字符串
     * @param isWin      是否 Windows 平台，Linux 需要对加密字符串进行 Base64 计算（如果字符串中有汉字或 url 特殊字符，需要设置）
     * @return 指定平台加密值
     */
    public static String encrypt(String privateKey, String plainText, boolean isWin) throws Exception {
        if (privateKey == null) privateKey = MY_PRIVATE_KEY_STRING;
        byte[] keyBytes = Base64.decodeBase64(privateKey);
        if (isWin) return encrypt(keyBytes, plainText);
        else return encryptByBase64(keyBytes, plainText);
    }

    /**
     * 解码 PublicKey
     *
     * @param key
     * @return
     */
    public static PublicKey getPublicKey(String key) {
        if (key == null || key.length() == 0) key = MY_PUBLIC_KEY_STRING;
        try {
            byte[] publicKeyBytes = Base64.decodeBase64(key);
            X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(publicKeyBytes);
            KeyFactory keyFactory = KeyFactory.getInstance("RSA", "SunRsaSign");
            return keyFactory.generatePublic(x509KeySpec);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 解码 PrivateKey
     *
     * @param key
     * @return
     */
    public static PrivateKey getPrivateKey(String key) {
        if (key == null || key.length() == 0) key = MY_PRIVATE_KEY_STRING;
        try {
            byte[] byteKey = Base64.decodeBase64(key);
            PKCS8EncodedKeySpec x509EncodedKeySpec = new PKCS8EncodedKeySpec(byteKey);
```

```
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            return keyFactory.generatePrivate(x509EncodedKeySpec);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    private static String decryptByBase64(PublicKey publicKey, String cipherText) throws
Exception {
        if (cipherText == null || cipherText.length() == 0) return cipherText;
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        try {
            cipher.init(Cipher.DECRYPT_MODE, publicKey);
        } catch (InvalidKeyException e) {
            // 因为 IBM JDK 不支持私钥加密，公钥解密，所以要反转公私钥
            // 也就是说对于解密，可以通过公钥的参数伪造一个私钥对象欺骗 IBM JDK
            RSAPublicKey rsaPublicKey = (RSAPublicKey) publicKey;
            RSAPrivateKeySpec spec = new RSAPrivateKeySpec(rsaPublicKey.getModulus(),
rsaPublicKey.getPublicExponent());
            Key fakePrivateKey = KeyFactory.getInstance("RSA").generatePrivate(spec);
            cipher = Cipher.getInstance("RSA"); //It is a stateful object. so we need to get
new one.
            cipher.init(Cipher.DECRYPT_MODE, fakePrivateKey);
        }

        // 开始分段解密
        byte[] encryptedData = Base64.decodeBase64(cipherText);
        int inputLen = encryptedData.length;
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        int offSet = 0;
        byte[] cache;
        int i = 0;
        // 对数据分段解密
        while (inputLen - offSet > 0) {
            if (inputLen - offSet > MAX_DECRYPT_BLOCK)
                cache = cipher.doFinal(encryptedData, offSet, MAX_DECRYPT_BLOCK);
            else
                cache = cipher.doFinal(encryptedData, offSet, inputLen - offSet);
            out.write(cache, 0, cache.length);
            i++;
            offSet = i * MAX_DECRYPT_BLOCK;
        }
        byte[] decryptedData = out.toByteArray();
```

```
        out.close();
        // 结束分段加密
        return new String(Base64.decodeBase64(decryptedData), "utf-8");
    }

    private static String encryptByBase64(byte[] keyBytes, String plainText) throws Exception
{
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory factory = KeyFactory.getInstance("RSA", "SunRsaSign");
        PrivateKey privateKey = factory.generatePrivate(spec);
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        try {
            cipher.init(Cipher.ENCRYPT_MODE, privateKey);
        } catch (InvalidKeyException e) {
            //For IBM JDK, 原因请看解密方法中的说明
            RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) privateKey;
            RSAPublicKeySpec publicKeySpec = new
RSAPublicKeySpec(rsaPrivateKey.getModulus(),
                    rsaPrivateKey.getPrivateExponent());
            Key fakePublicKey =
KeyFactory.getInstance("RSA").generatePublic(publicKeySpec);
            cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, fakePublicKey);
        }
        // 开始分段处理
        byte[] data = Base64.encodeBase64URLSafe(plainText.getBytes("utf-8"));
        int inputLen = data.length;
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        int offSet = 0;
        byte[] cache;
        int i = 0;
        while (inputLen - offSet > 0) {
            if (inputLen - offSet > MAX_ENCRYPT_BLOCK)
                cache = cipher.doFinal(data, offSet, MAX_ENCRYPT_BLOCK);
            else
                cache = cipher.doFinal(data, offSet, inputLen - offSet);
            out.write(cache, 0, cache.length);
            i++;
            offSet = i * MAX_ENCRYPT_BLOCK;
        }
        byte[] encryptedData = out.toByteArray();
        out.close();
        // 结束分段处理
        //byte[] encryptedBytes = cipher.doFinal(plainText.getBytes("UTF-8"));
```

```
        //String encryptedString = Base64.byteArrayToBase64(encryptedBytes);
        String encryptedString = Base64.encodeBase64String(encryptedData);
        return encryptedString;
    }

    private static String decrypt(PublicKey publicKey, String cipherText) throws Exception
{
        if (cipherText == null || cipherText.length() == 0) return cipherText;
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        try {
            cipher.init(Cipher.DECRYPT_MODE, publicKey);
        } catch (InvalidKeyException e) {
            // 因为 IBM JDK 不支持私钥加密，公钥解密，所以要反转公私钥
            // 也就是说对于解密，可以通过公钥的参数伪造一个私钥对象欺骗 IBM JDK
            RSAPublicKey rsaPublicKey = (RSAPublicKey) publicKey;
            RSAPrivateKeySpec spec = new RSAPrivateKeySpec(rsaPublicKey.getModulus(),
rsaPublicKey.getPublicExponent());
            Key fakePrivateKey = KeyFactory.getInstance("RSA").generatePrivate(spec);
            cipher = Cipher.getInstance("RSA"); //It is a stateful object. so we need to get
new one.
            cipher.init(Cipher.DECRYPT_MODE, fakePrivateKey);
        }

        //byte[] cipherBytes = Base64.base64ToByteArray(cipherText);
        //byte[] plainBytes = cipher.doFinal(cipherBytes);

        // 开始分段解密
        byte[] encryptedData = Base64.decodeBase64(cipherText);
        int inputLen = encryptedData.length;
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        int offSet = 0;
        byte[] cache;
        int i = 0;
        // 对数据分段解密
        while (inputLen - offSet > 0) {
            if (inputLen - offSet > MAX_DECRYPT_BLOCK)
                cache = cipher.doFinal(encryptedData, offSet, MAX_DECRYPT_BLOCK);
            else
                cache = cipher.doFinal(encryptedData, offSet, inputLen - offSet);
            out.write(cache, 0, cache.length);
            i++;
            offSet = i * MAX_DECRYPT_BLOCK;
        }
        byte[] decryptedData = out.toByteArray();
```

```
        out.close();
        // 结束分段加密
        // return new String(plainBytes);
        return new String(decryptedData);
    }

    private static String encrypt(byte[] keyBytes, String plainText) throws Exception {
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory factory = KeyFactory.getInstance("RSA", "SunRsaSign");
        PrivateKey privateKey = factory.generatePrivate(spec);
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        try {
            cipher.init(Cipher.ENCRYPT_MODE, privateKey);
        } catch (InvalidKeyException e) {
            //For IBM JDK, 原因请看解密方法中的说明
            RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) privateKey;
            RSAPublicKeySpec publicKeySpec = new
RSAPublicKeySpec(rsaPrivateKey.getModulus(),
                    rsaPrivateKey.getPrivateExponent());
            Key fakePublicKey =
KeyFactory.getInstance("RSA").generatePublic(publicKeySpec);
            cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, fakePublicKey);
        }
        // 开始分段处理
        byte[] data = plainText.getBytes("UTF-8");
        int inputLen = data.length;
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        int offSet = 0;
        byte[] cache;
        int i = 0;
        while (inputLen - offSet > 0) {
            if (inputLen - offSet > MAX_ENCRYPT_BLOCK)
                cache = cipher.doFinal(data, offSet, MAX_ENCRYPT_BLOCK);
            else
                cache = cipher.doFinal(data, offSet, inputLen - offSet);
            out.write(cache, 0, cache.length);
            i++;
            offSet = i * MAX_ENCRYPT_BLOCK;
        }
        byte[] encryptedData = out.toByteArray();
        out.close();
        // 结束分段处理
        //byte[] encryptedBytes = cipher.doFinal(plainText.getBytes("UTF-8"));
```

```
        //String encryptedString = Base64.byteArrayToBase64(encryptedBytes);
        String encryptedString = Base64.encodeBase64String(encryptedData);
        return encryptedString;
    }

    public static void main(String[] args) throws Exception {
        initKey();
        String in = "[{a1:哇哈哈,a2:kdscvh,a3:12345678}";
        //in = Base64.encodeBase64URLSafeString(in.getBytes("UTF-8"));
        String text1 = RSAUtil.encrypt(MY_PRIVATE_KEY_STRING, in, true);
        System.out.println("in:" + text1);
        String text2 = RSAUtil.decrypt(MY_PUBLIC_KEY_STRING, text1, true);
        System.out.println("out:" + java.net.URLDecoder.decode(text2, "utf-8"));
    }
}
```

2.2. 生成公钥/密钥

通过 RSA 加密解密算法生成公钥/密钥对，接口提供方与接口调用方都使用此类生成公钥/密钥对，并向对方（国家）公布公钥，在请求时使用己方密钥进行加密，获得返回结果时使用对端公钥进行解密。

```
import java.security.Key;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.HashMap;
import java.util.Map;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;

public class CreateSecretKey {
    public static final String KEY_ALGORITHM = "RSA";
    public static final String SIGNATURE_ALGORITHM="MD5withRSA";
    private static final String PUBLIC_KEY = "RSAPublicKey";
    private static final String PRIVATE_KEY = "RSAPrivateKey";
```

```
    /**
     * RSA 最大加密明文大小
     */
    private static final int MAX_ENCRYPT_BLOCK = 117;

    /**
     * RSA 最大解密密文大小
     */
    private static final int MAX_DECRYPT_BLOCK = 128;
    //获得公钥字符串
    public static String getPublicKeyStr(Map<String, Object> keyMap) throws Exception {
        //获得 map 中的公钥对象 转为 key 对象
        Key key = (Key) keyMap.get(PUBLIC_KEY);
        //编码返回字符串
        return encryptBASE64(key.getEncoded());
    }

    public static Map<String, Object> initKey() throws Exception {
        KeyPairGenerator keyPairGen = KeyPairGenerator
                .getInstance(KEY_ALGORITHM);
        keyPairGen.initialize(1024);
        KeyPair keyPair = keyPairGen.generateKeyPair();
        RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
        RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
        Map<String, Object> keyMap = new HashMap<String, Object>(2);
        keyMap.put(PUBLIC_KEY, publicKey);
        keyMap.put(PRIVATE_KEY, privateKey);
        return keyMap;
    }

    //获得私钥字符串
    public static String getPrivateKeyStr(Map<String, Object> keyMap) throws Exception {
        //获得 map 中的私钥对象 转为 key 对象
        Key key = (Key) keyMap.get(PRIVATE_KEY);
        //编码返回字符串
        return encryptBASE64(key.getEncoded());
    }

    //获取公钥
    public static PublicKey getPublicKey(String key) throws Exception {
        byte[] keyBytes;
        keyBytes = (new BASE64Decoder()).decodeBuffer(key);
        X509EncodedKeySpec keySpec = new X509EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
```

```
        PublicKey publicKey = keyFactory.generatePublic(keySpec);
        return publicKey;
    }

    //获取私钥
    public static PrivateKey getPrivateKey(String key) throws Exception {
        byte[] keyBytes;
        keyBytes = (new BASE64Decoder()).decodeBuffer(key);
        PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        PrivateKey privateKey = keyFactory.generatePrivate(keySpec);
        return privateKey;
    }

    //解码返回 byte
    public static byte[] decryptBASE64(String key) throws Exception {
        return (new BASE64Decoder()).decodeBuffer(key);
    }

    //编码返回字符串
    public static String encryptBASE64(byte[] key) throws Exception {
        return (new BASE64Encoder()).encodeBuffer(key);
    }

    public static void main(String[] args) {
        Map<String, Object> keyMap;
        byte[] cipherText;
        String input = "Hello World!";
        try {
            keyMap = initKey();
            String publicKey = getPublicKeyStr(keyMap);
            System.out.println("公钥------------------");
            System.out.println(publicKey);
            String privateKey = getPrivateKeyStr(keyMap);
            System.out.println("私钥------------------");
            System.out.println(privateKey);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.3. 生成用户签名

通过 RSA 加密生成，用户签名为请求方用户名+密码+时间戳 Json 字符串用 RSA 加密算法使用己方密钥加密获得，通过请求头将签名发送到提供方。

提供方解析请求头获得加密的用户签名，用请求方提供的公钥将用户签名解密，匹配解密后的用户、密码，验证用户访问接口的权限。

3. 性能要求

1-8 秒内能返回数据。

三、接口目录明细

4. 资源目录数据统计接口（TJ_ZYML）

资源提供方提供独立接口，提供资源目录中数据数据量，以及信息资源情况说明。

4.1. 资源目录数据统计（getDataCatalogsInfo）

请求头（Header）：

	参数	参数名称	是否为空	类型	长度	说明
1	appsign	请求签名	否	字符串		根据请求方用户名+密码+时间戳Json字符串用RSA加密得到

请求头 appsign 的解密 Json 格式如下：

```
{
  " username":" gjxyzx",//用户名
  " password":"xxxx",//密码
  " timespamp":"xxxx",//时间戳
}
```

请求参数（根据不同部门接口不同）：

	参数	参数名称	是否为空	类型	长度	说明
1	reqparams	请求字符串		字符串		加密后的 json 格式的请求字符串。

请求参数 reqparams 的解密 Json 格式如下：

```
{
  " servicename":"0"   //资源目录名称
}
```

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	servicename	资源目录名称	字符串	必填，填"0"为所有目录

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	资源目录名称	SERVICENAME	C(字符型)	200	必填
2	资源目录编码	SERVICECODE	C(字符型)	20	必填
3	资源目录说明	SERVICEDESC	C(字符型)	500	必填
4	数据量	RECORDNUM	N(数值型)	(19,0)	必填
5	数据资源说明	DATADESC	C(字符型)	500	必填
6	数据覆盖范围	SCOPE	C(字符型)	240	必填。全省或已覆盖的地市列表，用“，”分割

4.2. 接口响应处理/返回值生成示例

服务端响应接口后，得到请求信息，对请求信息进行解析、解密，进行业务处理，得到返回值后，再封装成 json 对象进行加密，加密后的 json 对象发送出去。

```
import com.alibaba.fastjson.JSONObject;
import javax.ws.rs.*;
import javax.ws.rs.core.Response;
import utils.RSAUtil;

@Path("/rscount")
```

```
public class ResourceCountImpl implements ResourceCount {
    public static String YU_PUBLIC_KEY_STRING = ""; //对端公钥
    private static String MY_PRIVATE_KEY_STRING = ""; //己方私钥
    private String username = "";
    private String password = "";

    public Response postCount(String reqparams1) {

        String reqheaders = ""; //todo 获得请求头参数
        String reqparams = ""; //todo 获得请求体参数
        String strDec = "", sVal = "", strRet = "", username = "", password = "", timespamp
= "";
        boolean isVerify = false; //请求参数校验标识
        JSONObject jresponse = new JSONObject(); //响应值

        //todo 从文件中获得有效的对端公钥、己方密钥
        MY_PRIVATE_KEY_STRING = "MIIC......ZdDA==";
        YU_PUBLIC_KEY_STRING = "MIGf......AQAB ";

        try {
            JSONObject jheaders = JSONObject.parseObject(reqheaders); //解读请求头
            sVal = (String) jheaders.get("appsign");
            strDec = RSAUtil.decrypt(YU_PUBLIC_KEY_STRING, sVal); //进行 RSA 解密
            JSONObject juser = JSONObject.parseObject(strDec); //解读解密后的请求头
            username = (String) juser.get("username");
            password = (String) juser.get("password");
            timespamp = (String) juser.get("timespamp");
            //todo 用户名、密码、时间戳验证，验证通过设置 isVerify=true;
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (isVerify) {
            JSONObject jparams = JSONObject.parseObject(reqparams); //解读请求参数
            sVal = (String) jparams.get("reqparams");
            try {
                strDec = RSAUtil.decrypt(YU_PUBLIC_KEY_STRING, sVal); //进行 RSA 解密
                JSONObject jparam = JSONObject.parseObject(strDec); //解读解密后的请求参数
                String servicename = (String) jparam.get("servicename");
                //todo 用 servicename 获取资源目录的统计记录数，获取成功设置 isVerify=true;
            } catch (Exception e) {
                e.printStackTrace();
            }
            if (isVerify) {
                //todo 生成成功响应 jresponse 对象;
```

```
            } else {
                //todo 生成"获取数据失败！"错误信息对象;
            }
        } else {
            //todo 生成"输入参数不正确！"错误信息对象;
        }
        jresponse=new JSONObject();
        try {
            strRet = RSAUtil.encrypt(MY_PRIVATE_KEY_STRING,
JSONObject.toJSONString(jresponse)); //返回值 RSA 解加密
        } catch (Exception e) {
            e.printStackTrace();
        }
        jresponse.put("resinfo",strRet);
        return Response.status(200).entity(jresponse).build();
    }
}
```

5. 法人公用事业信息归集目录（FR_GYSYML）

5.1. 法人用水缴费信息接口（getFrysjfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	tyshxydm	统一社会信用代码	字符串	必填
2	qymc	企业名称	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	企业名称	QYMC	C(字符型)	255	在工商、民政等部门登记的单位全称
2	统一社会信用代码	TYSHXYDM	C(字符型)	30	
3	缴费类型名	JFLXM	C(字符型)	100	
4	是否已缴费	SFYJF	C(字符型)	10	
5	计量开始月或日	JLKSYHR	D(日期型)		
6	计量截止月或日	JLJZYHR	D(日期型)		
7	使用量	SYL	N(数值型)	(10,0)	单位：吨
8	单位	SYDW	C(字符型)	255	
9	交费金额	JFJE	N(数值型)	(19,0)	单位：元
10	经办机构名称	JBJGQC	C(字符型)	255	

序号	信息项名称	数据标识	数据类型	数据长度	备注
11	经办机构统一社会信用代码	JBJGTYSHXYDM	C(字符型)	30	
12	缴纳日期	JNRQ	D(日期型)		

接口响应处理代码示例：

```
.....
    JSONObject jparams = JSONObject.parseObject(reqparams); //解读请求参数
    sVal = (String) jparams.get("reqparams");
    try {
        strDec = RSAUtil.decrypt(YU_PUBLIC_KEY_STRING, sVal); //进行 RSA 解密
        JSONObject jparam = JSONObject.parseObject(strDec); //解读解密后的请求参数
        String tyshxydm = (String) jparam.get("tyshxydm");
        //todo 用 tyshxydm 获取法人纳税人信息记录数，获取成功设置 isVerify=true;
    } catch (Exception e) {
        e.printStackTrace();
    }
    ......
```

5.2. 法人用水拖欠费信息接口（getFrysqfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	tyshxydm	统一社会信用代码	字符串	必填
2	qymc	企业名称	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	企业名称	QYMC	C(字符型)	255	在工商、民政等部门登记的单位全称
2	统一社会信用代码	TYSHXYDM	C(字符型)	30	
3	欠缴类型名称	QJLXMC	C(字符型)	100	
4	拖缴欠缴开始日期	QJTJKSRQ	D(日期型)		
5	拖缴欠缴截止日期	QJTJJZRQ	D(日期型)		
6	欠费金额	QFJE	N(数值型)	(19,0)	单位：元
7	认定机构	RDJG	C(字符型)	255	
8	认定机构统一代码	RDJGTYSHXYDM	C(字符型)	30	
9	认定日期	RDRQ	D(日期型)		

5.3. 法人用电缴费信息接口（getFrydjfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	tyshxydm	统一社会信用代码	字符串	必填
2	qymc	企业名称	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	企业名称	QYMC	C(字符型)	255	在工商、民政等部门登记的单位全称
2	统一社会信用代码	TYSHXYDM	C(字符型)	30	
3	缴费类型名	JFLXM	C(字符型)	100	
4	是否已缴费	SFYJF	C(字符型)	10	
5	计量开始月或日	JLKSYHR	D(日期型)		
6	计量截止月或日	JLJZYHR	D(日期型)		
7	使用量	SYL	N(数值型)	(10,0)	单位：度
8	使用单位	SYDW	C(字符型)	255	
9	交费金额	JFJE	N(数值型)	(19,0)	单位：元
10	经办机构名称	JBJGQC	C(字符型)	255	
11	经办机构统一社会信用代码	JBJGTYSHXYDM	C(字符型)	30	
12	缴纳日期	JNRQ	D(日期型)		

5.4. 法人用电拖欠费信息接口（getFrydqfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	tyshxydm	统一社会信用代码	字符串	必填
2	qymc	企业名称	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	企业名称	QYMC	C(字符型)	255	在工商、民政等部门登记的单位全称
2	统一社会信用代码	TYSHXYDM	C(字符型)	30	
3	欠缴类型名称	QJLXMC	C(字符型)	100	
4	拖缴欠缴开始日期	QJTJKSRQ	D(日期型)		
5	拖缴欠缴截止日期	QJTJJZRQ	D(日期型)		

序号	信息项名称	数据标识	数据类型	数据长度	备注
6	欠费金额	QFJE	N(数值型)	(19,0)	单位：元
7	认定机构	RDJG	C(字符型)	255	
8	认定机构统一代码	RDJGTYSHXYDM	C(字符型)	30	
9	认定日期	RDRQ	D(日期型)		

5.5. 法人用气缴费信息接口（getFryqjfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	tyshxydm	统一社会信用代码	字符串	必填
2	qymc	企业名称	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	企业名称	QYMC	C(字符型)	255	在工商、民政等部门登记的单位全称
2	统一社会信用代码	TYSHXYDM	C(字符型)	30	
3	缴费类型名	JFLXM	C(字符型)	100	
4	是否已缴费	SFYJF	C(字符型)	10	
5	计量开始月或日	JLKSYHR	D(日期型)		
6	计量截止月或日	JLJZYHR	D(日期型)		
7	使用量	SYL	N(数值型)	(10,0)	单位：立方米
8	单位	SYDW	C(字符型)	255	
9	交费金额	JFJE	N(数值型)	(19,0)	单位：元
10	经办机构名称	JBJGQC	C(字符型)	255	
11	经办机构统一社会信用代码	JBJGTYSHXYDM	C(字符型)	30	
12	缴纳日期	JNRQ	D(日期型)		

5.6. 法人用气拖欠费信息接口（getFryqqfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	tyshxydm	统一社会信用代码	字符串	必填
2	qymc	企业名称	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	企业名称	QYMC	C(字符型)	255	在工商、民政等部门登记的单位全称
2	统一社会信用代码	TYSHXYDM	C(字符型)	30	
3	欠缴类型名称	QJLXMC	C(字符型)	255	
4	拖缴欠缴开始日期	QJTJKSRQ	D(日期型)		
5	拖缴欠缴截止日期	QJTJJZRQ	D(日期型)		
6	欠费金额	QFJE	N(数值型)	(19,0)	单位：元
7	认定机构	RDJG	C(字符型)	255	
8	认定机构统一代码	RDJGTYSHXYDM	C(字符型)	30	
9	认定日期	RDRQ	D(日期型)		

5.7. 仓储物流寄件量信息接口（getCcwljjlxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	tyshxydm	统一社会信用代码	字符串	必填
2	qymc	企业名称	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	公司名称	QYMC	C(字符型)	255	在工商、民政等部门登记的单位全称
2	统一社会信用代码	TYSHXYDM	C(字符型)	30	
3	月份	YF	C(字符型)	50	格式: YYYYMM
4	寄件量	JJL	N(数值型)	(10,0)	
5	总费用	ZFY	N(数值型)	(19,0)	单位：元
6	运费	YF	N(数值型)	(19,0)	单位：元

5.8. 仓储物流逾期信息接口（getCcwlyqxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	tyshxydm	统一社会信用代码	字符串	必填
2	qymc	企业名称	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	公司名称	QYMC	C(字符型)	255	在工商、民政等部门登记的单位全称
2	统一社会信用代码	TYSHXYDM	C(字符型)	30	
3	逾期年度	YQND	C(字符型)	50	格式: YYYY
4	逾期次数	YQCS	N(数值型)	(10,0)	
5	最大逾期金额	ZDYQJE	N(数值型)	(19,0)	单位：元
6	最长逾期天数	ZDYQTS	N(数值型)	(10,0)	

6. 自然人公用事业信息归集目录（ZRR_ZYGYSYML）

6.1. 自然人用水缴费信息接口（getZrrysjfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	zjlx	证件类型	字符串	必填
2	zjhm	证件号码	字符串	必填

接口返回信息：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	姓名	XM	C(字符型)	255	
2	身份证件类型	SFZJLX	C(字符型)	50	
3	身份证件号码	SFZJHM	C(字符型)	30	
4	缴费类型名称	JFLXMC	C(字符型)	100	
5	是否已缴费	SFYJF	C(字符型)	10	
6	计量开始月或日	JLKSYHR	D(日期型)		
7	计量截止月或日	JLJZYHR	D(日期型)		
8	使用量	SYL	N(数值型)	(10,0)	单位：吨
9	单位	SYDW	C(字符型)	255	
10	交费金额	JFJE	N(数值型)	(19,0)	单位：元
11	经办机构名称	JBJGQC	C(字符型)	255	
12	经办机构统一社会信用代码	JBJGTYSHXYDM	C(字符型)	30	
13	缴纳日期	JNRQ	D(日期型)		

接口响应处理代码示例：

```
.....
```

```
JSONObject jparams = JSONObject.parseObject(reqparams); //解读请求参数
sVal = (String) jparams.get("reqparams");
try {
    strDec = RSAUtil.decrypt(YU_PUBLIC_KEY_STRING, sVal); //进行 RSA 解密
    JSONObject jparam = JSONObject.parseObject(strDec); //解读解密后的请求参数
    String zjlx = (String) jparam.get("zjlx");
    String zjhm = (String) jparam.get("zjhm");
    //todo 用 zjlx、zjhm 获取自然人纳税人信息记录数，获取成功设置 isVerify=true;
} catch (Exception e) {
    e.printStackTrace();
}
......
```

6.2. 自然人用水拖欠费信息接口（getZrrysqfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	zjlx	证件类型	字符串	必填
2	zjhm	证件号码	字符串	必填

接口返回值：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	姓名	XM	C(字符型)	255	
2	身份证件类型	SFZJLX	C(字符型)	50	
3	身份证件号码	SFZJHM	C(字符型)	30	
4	欠缴类型名称	QJLXMC	C(字符型)	100	
5	欠费开始日期	QJTJKSRQ	D(日期型)		
6	欠费截止日期	QJTJJZRQ	D(日期型)		
7	欠费金额	QFJE	N(数值型)	(19,0)	单位：元
8	认定机构	RDJG	C(字符型)	255	
9	认定机构统一代码	RDJGTYSHXYDM	C(字符型)	30	
10	认定日期	RDRQ	D(日期型)		

6.3. 自然人用电缴费信息接口（getZrrydjfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	zjlx	证件类型	字符串	必填
2	zjhm	证件号码	字符串	必填

接口返回值：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	姓名	XM	C(字符型)	255	
2	身份证件类型	SFZJLX	C(字符型)	50	
3	身份证件号码	SFZJHM	C(字符型)	30	
4	缴费类型名称	JFLXMC	C(字符型)	100	
5	是否已缴费	SFYJF	C(字符型)	10	
6	计量开始月或日	JLKSYHR	D(日期型)		
7	计量截止月或日	JLJZYHR	D(日期型)		
8	使用量	SYL	N(数值型)	(10,0)	单位：度
9	使用单位	SYDW	C(字符型)	255	
10	交费金额	JFJE	N(数值型)	(19,0)	单位：元
11	经办机构名称	JBJGQC	C(字符型)	255	
12	经办机构统一社会信用代码	JBJGTYSHXYDM	C(字符型)	30	
13	缴纳日期	JNRQ	D(日期型)		

6.4. 自然人用电拖欠费信息接口（getZrrydqfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	zjlx	证件类型	字符串	必填
2	zjhm	证件号码	字符串	必填

接口返回值：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	姓名	XM	C(字符型)	255	
2	身份证件类型	SFZJLX	C(字符型)	50	
3	身份证件号码	SFZJHM	C(字符型)	30	
4	欠缴类型名称	QJLXMC	C(字符型)	100	
5	欠费开始时间	QJTJKSRQ	D(日期型)		
6	欠费截止日期	QJTJJZRQ	D(日期型)		
7	欠费金额	QFJE	N(数值型)	(19,0)	单位：元
8	认定机构	RDJG	C(字符型)	255	
9	认定机构统一代码	RDJGTYSHXYDM	C(字符型)	30	
10	认定日期	RDRQ	D(日期型)		

6.5. 自然人用气缴费信息接口（getZrryqjfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	zjlx	证件类型	字符串	必填
2	zjhm	证件号码	字符串	必填

接口返回值：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	姓名	XM	C(字符型)	255	
2	身份证件类型	SFZJLX	C(字符型)	50	
3	身份证件号码	SFZJHM	C(字符型)	30	
4	缴费类型名称	JFLXMC	C(字符型)	100	
5	是否已缴费	SFYJF	C(字符型)	10	
6	计量开始月或日	JLKSYHR	D(日期型)		
7	计量截止月或日	JLJZYHR	D(日期型)		
8	使用量	SYL	N(数值型)	(10,0)	单位：立方米
9	单位	SYDW	C(字符型)	255	
10	交费金额	JFJE	N(数值型)	(19,0)	单位：元
11	经办机构名称	JBJGQC	C(字符型)	255	
12	经办机构统一社会信用代码	JBJGTYSHXYDM	C(字符型)	30	
13	缴纳日期	JNRQ	D(日期型)		

6.6. 自然人用气拖欠费信息接口（getZrryqqfxx）

reqparams 参数说明：

序号	参数 ID	参数名称	类型	备注
1	zjlx	证件类型	字符串	必填
2	zjhm	证件号码	字符串	必填

接口返回值：

序号	信息项名称	数据标识	数据类型	数据长度	备注
1	姓名	XM	C(字符型)	255	
2	身份证件类型	SFZJLX	C(字符型)	50	
3	身份证件号码	SFZJHM	C(字符型)	30	
4	欠缴类型名称	QJLXMC	C(字符型)	100	
5	欠费开始时间	QJTJKSRQ	D(日期型)		
6	欠费截止日期	QJTJJZRQ	D(日期型)		
7	欠费金额	QFJE	N(数值型)	(19,0)	单位：元

序号	信息项名称	数据标识	数据类型	数据长度	备注
8	认定机构	RDJG	C(字符型)	255	
9	认定机构统一代码	RDJGTYSHXYDM	C(字符型)	30	
10	认定日期	RDRQ	D(日期型)		

四、接口登记表

序号	提供方	接口地址	接口名称	公钥	备注
1					
2					