RSA 签名 / 验签工具

作者：hanzanr123
原文链接：https://ld246.com/article/1553854312632
来源网站：链滴

依赖jar包下载

zmxysdkjava20170605134301.jar

zmxysdkjava20170605134301source.jar

代码如下

```
  import com.antgroup.zmxy.openplatform.api.ZhimaApiException;
  import com.antgroup.zmxy.openplatform.api.internal.util.Base64Util;
  import com.antgroup.zmxy.openplatform.api.internal.util.CoderUtil;
  import com.antgroup.zmxy.openplatform.api.internal.util.EncryptionModeEnum;
  import com.antgroup.zmxy.openplatform.api.internal.util.SignTypeEnum;
  import com.antgroup.zmxy.openplatform.api.internal.util.json.ExceptionErrorListener;
  import com.antgroup.zmxy.openplatform.api.internal.util.json.JSONValidatingReader;
   import java.io.ByteArrayOutputStream;
   import java.security.Key;
   import java.security.KeyFactory;
   import java.security.PrivateKey;
   import java.security.PublicKey;
   import java.security.Signature;
   import java.security.spec.PKCS8EncodedKeySpec;
   import java.security.spec.RSAPrivateKeySpec;
   import java.security.spec.RSAPublicKeySpec;
   import java.security.spec.X509EncodedKeySpec;
   import java.util.Iterator;
   import java.util.Map;
   import javax.crypto.Cipher;

  import org.apache.log4j.Logger;

   public class PKRSACoderUtil extends CoderUtil{

     protected static Logger log = Logger.getLogger(PKRSACoderUtil.class);

     public static final String KEY_ALGORTHM = "RSA";
     public static final String SPECIFIC_KEY_ALGORITHM = "RSA/ECB/PKCS1Padding";
     public static final String SIGNATURE_ALGORITHM = "SHA256WITHRSA";
     public static String encrypt(String paramsString, String charset, String publicKey)
   throws Exception
     {
       byte[] encryptedResult = encryptByPublicKey(paramsString.getBytes(charset), publicKey,
null);
       return Base64Util.byteArrayToBase64(encryptedResult);
     }
    public static String encrypt(String paramsString, String charset, String publicKey, Encryptio
ModeEnum encryptionType)
   throws Exception
     {
       byte[] encryptedResult = encryptByPublicKey(paramsString.getBytes(charset), publicKey,
encryptionType);


       return Base64Util.byteArrayToBase64(encryptedResult);
```

```
    }


    public static String sign(String data, String charset, String privateKey)
  throws Exception
    {
      byte[] dataInBytes = data.getBytes(charset);
      String signParams = sign(dataInBytes, privateKey);
      return signParams;
    }



    public static String sign(SignTypeEnum signType, String data, String charset, String privat
Key)
  throws Exception
    {
      byte[] dataInBytes = data.getBytes(charset);
      String signParams = sign(signType, dataInBytes, privateKey);
      return signParams;
    }

    public static String decrypt(String data, String key, String charset)
  throws Exception
    {
      byte[] byte64 = Base64Util.base64ToByteArray(data);
      byte[] encryptedBytes = decryptByPrivateKey(byte64, key, null);
      return new String(encryptedBytes, charset);
    }


    public static String decrypt(String data, String key, String charset, EncryptionModeEnum e
cryptionType)
  throws Exception
    {
      byte[] byte64 = Base64Util.base64ToByteArray(data);
      byte[] encryptedBytes = decryptByPrivateKey(byte64, key, encryptionType);
      return new String(encryptedBytes, charset);
    }


    public static byte[] decryptByPrivateKey(byte[] data, String key, EncryptionModeEnum enc
yptionType)
  throws Exception
    {
      byte[] decryptedData = null;


      byte[] keyBytes = decryptBASE64(key);

      PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(keyBytes);
      KeyFactory keyFactory = KeyFactory.getInstance("RSA");
      Key privateKey = keyFactory.generatePrivate(pkcs8EncodedKeySpec);
```

```
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
   cipher.init(2, privateKey);

    int maxDecryptBlockSize;
    if (encryptionType != null)
      maxDecryptBlockSize = getMaxDecryptBlockSizeByEncryptionType(encryptionType);
    else {
      maxDecryptBlockSize = getMaxDecryptBlockSize(keyFactory, privateKey);
    }

    ByteArrayOutputStream bout = new ByteArrayOutputStream();
    try {
      int dataLength = data.length;
      for (int i = 0; i < dataLength; i += maxDecryptBlockSize) {
        int decryptLength = (dataLength - i < maxDecryptBlockSize) ? dataLength - i : maxDec
yptBlockSize;

        byte[] doFinal = cipher.doFinal(data, i, decryptLength);
        bout.write(doFinal);
      }
      decryptedData = bout.toByteArray();
    } finally {
      if (bout != null) {
        bout.close();
      }
    }

    return decryptedData;
  }


   public static byte[] encryptByPublicKey(byte[] data, String key, EncryptionModeEnum encr
ptionType)
  throws Exception
  {
    byte[] encryptedData = null;


    byte[] keyBytes = decryptBASE64(key);

    X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(keyBytes);
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    Key publicKey = keyFactory.generatePublic(x509EncodedKeySpec);


    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(1, publicKey);

    int maxEncryptBlockSize;
    if (encryptionType != null)
      maxEncryptBlockSize = getMaxEncryptBlockSizeByEncryptionType(encryptionType);
    else {
      maxEncryptBlockSize = getMaxEncryptBlockSize(keyFactory, publicKey);
    }
```

```
      ByteArrayOutputStream bout = new ByteArrayOutputStream();
      try {
        int dataLength = data.length;
        for (int i = 0; i < data.length; i += maxEncryptBlockSize) {
          int encryptLength = (dataLength - i < maxEncryptBlockSize) ? dataLength - i : maxEncr
ptBlockSize;

          byte[] doFinal = cipher.doFinal(data, i, encryptLength);
          bout.write(doFinal);
        }
        encryptedData = bout.toByteArray();
      } finally {
        if (bout != null) {
          bout.close();
        }
      }
      return encryptedData;
    }

    public static String sign(byte[] data, String privateKey)
  throws Exception
    {
      return sign(SignTypeEnum.SHA1WITHRSA, data, privateKey);
    }


    public static String sign(SignTypeEnum signType, byte[] data, String privateKey)
  throws Exception
    {
      byte[] keyBytes = decryptBASE64(privateKey);

      PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(keyBytes);

      KeyFactory keyFactory = KeyFactory.getInstance("RSA");

      PrivateKey privateKey2 = keyFactory.generatePrivate(pkcs8EncodedKeySpec);


      Signature signature = Signature.getInstance(signType.getDesc());
      signature.initSign(privateKey2);
      signature.update(data);

      return encryptBASE64(signature.sign());
    }


  public static boolean verify(byte[] data, String publicKey, String sign)
    throws Exception
  {
    return verify(SignTypeEnum.SHA1WITHRSA, data, publicKey, sign);
  }
```

```
  public static boolean verify(SignTypeEnum signType, byte[] data, String publicKey, String sig
)
    throws Exception
  {
    byte[] keyBytes = decryptBASE64(publicKey);

    X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(keyBytes);

    KeyFactory keyFactory = KeyFactory.getInstance("RSA");

    PublicKey publicKey2 = keyFactory.generatePublic(x509EncodedKeySpec);

    Signature signature = Signature.getInstance(signType.getDesc());
    signature.initVerify(publicKey2);
    signature.update(data);
    return signature.verify(decryptBASE64(sign));
  }



  public static String decryptResponse(String fullResponse, String privateKey, String charset, E
cryptionModeEnum encryptionType)
    throws Exception
  {
    String decryptedRsp = null;
    Map rootJson = parseResponseMap(fullResponse);
    for (Iterator it = rootJson.keySet().iterator(); it.hasNext(); ) {
      String key = (String)it.next();
      if (key.endsWith("_response")) {
        String value = (String)rootJson.get(key);
        decryptedRsp = value;
      }
    }

    if (((Boolean)rootJson.get("encrypted")).booleanValue()) {
      decryptedRsp = decrypt(decryptedRsp, privateKey, charset, encryptionType);
    }
    return decryptedRsp;
  }



  public static void verifySign(String fullResponse, String decryptedBizResponse, String public
ey, String charset)
    throws Exception
  {
    verifySign(SignTypeEnum.SHA1WITHRSA, fullResponse, decryptedBizResponse, publicKey,
harset);
  }



  public static void verifySign(SignTypeEnum signType, String fullResponse, String decryptedB
```

```
zResponse, String publicKey, String charset)
    throws Exception
  {
    Map rootJson = parseResponseMap(fullResponse);
    String sign = (String)rootJson.get("biz_response_sign");

    if ((sign != null) && (sign.length() > 0)) {
      boolean success = verify(signType, decryptedBizResponse.getBytes(charset), publicKey, si
n);

      if (!(success))
        throw new ZhimaApiException("验签失败: " + sign.toString());
    }
  }

  public static Map parseResponseMap(String fullResponse)
    throws ZhimaApiException
  {
    JSONValidatingReader reader = new JSONValidatingReader(new ExceptionErrorListener());
    Object rootObj = reader.read(fullResponse);
    if (rootObj instanceof Map) {
      Map rootJson = (Map)rootObj;
      return rootJson;
    }

    throw new ZhimaApiException("返回结果格式有误:" + fullResponse);
  }









  private static int getMaxEncryptBlockSize(KeyFactory keyFactory, Key key)
    throws Exception
  {
    int maxLength = 117;
    try {
      RSAPublicKeySpec publicKeySpec = (RSAPublicKeySpec)keyFactory.getKeySpec(key, RSAP
blicKeySpec.class);
      int keyLength = publicKeySpec.getModulus().bitLength();
      maxLength = keyLength / 8 - 11;
    }
    catch (Exception e) {
    }
    return maxLength;
  }
```

```
  private static int getMaxEncryptBlockSizeByEncryptionType(EncryptionModeEnum encryptio
Type)
  {
    if (encryptionType == EncryptionModeEnum.RSA1024)
      return 117;
    if (encryptionType == EncryptionModeEnum.RSA2048) {
      return 245;
    }

    return 117;
  }
```

```
  private static int getMaxDecryptBlockSize(KeyFactory keyFactory, Key key)
    throws Exception
  {
    int maxLength = 128;
    try {
      RSAPrivateKeySpec publicKeySpec = (RSAPrivateKeySpec)keyFactory.getKeySpec(key, RS
PrivateKeySpec.class);
      int keyLength = publicKeySpec.getModulus().bitLength();
      maxLength = keyLength / 8;
    }
    catch (Exception e) {
    }
    return maxLength;
  }
```

```
  private static int getMaxDecryptBlockSizeByEncryptionType(EncryptionModeEnum encryptio
Type)
  {
    if (encryptionType == EncryptionModeEnum.RSA1024)
      return 128;
    if (encryptionType == EncryptionModeEnum.RSA2048) {
      return 256;
    }

    return 128;
  }
}
```

验签代码

```
/**

 * 验证签名
 * @return
 * @throws Exception
 */
public static boolean checksign(JSONObject jsonObject,String platpublickey) throws Exceptio
{
    //获取签名
    String sign = jsonObject.getString("sign");
    //json对象转换成map
    Map<String,Object> bizParams = getTextParams(jsonObject);
    String content = unurlgetSignCheckContentV2(bizParams).trim().replace("\\/", "/");
    return PKRSACoderUtil.verify(SignTypeEnum.SHA256WITHRSA, content.getBytes(SysUtil.C
ARSET),platpublickey, sign);
}
```

签名代码

```
/**
 * 生成签名
 *
 *
 * @return
 *
 */
public static String producesignByJson(JSONObject jsonObject ,String zzrsprivatekey){
    String signstr="";
    try{
            String content = unurlgetSignCheckContentV2(getTextParams(jsonObject)).trim();
            signstr = PKRSACoderUtil.sign(SignTypeEnum.SHA256WITHRSA, content, SysUtil.CHA
SET, zzrsprivatekey);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return signstr;
}
```