RSA加密算法研究-伟鹏

**package** com.sunshine.utils;

**import** java.io.ByteArrayOutputStream;

**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.Signature;

**import** java.security.interfaces.RSAKey;

**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** javax.crypto.Cipher;

**import** sun.misc.BASE64Decoder;

**import** sun.misc.BASE64Encoder;

**public** **class** RSAEncryption {

**public** **static** **final** String ***KEY\_ALGORITHM*** = "RSA";

**private** **static** **final** String ***PUBLIC\_KEY*** = "RSAPublicKey";

**private** **static** **final** String ***PRIVATE\_KEY*** = "RSAPrivateKey";

**public** **static** **final** String ***SIGNATURE\_ALGORITHM***="MD5withRSA";

/\*\*

\* 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;

//初始化RSAKey

**public** **static** Map<String, RSAKey> 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,RSAKey> keyMap = **new** HashMap<String,RSAKey>(2);

keyMap.put(***PUBLIC\_KEY***, publicKey);

keyMap.put(***PRIVATE\_KEY***, privateKey);

**return** keyMap;

}

//获得公钥字符串

**public** **static** String getPublicKeyStr(Map<String, RSAKey> 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());

}

//获得私钥字符串

**public** **static** String getPrivateKeyStr(Map<String, RSAKey> 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);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*签名和验证\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **byte**[] sign(**byte**[] data,String privateKeyStr) **throws** Exception{

PrivateKey priK = *getPrivateKey*(privateKeyStr);

Signature sig = Signature.*getInstance*(***SIGNATURE\_ALGORITHM***);

sig.initSign(priK);

sig.update(data);

**return** sig.sign();

}

**public** **static** **boolean** verify(**byte**[] data,**byte**[] sign,String publicKeyStr) **throws** Exception{

PublicKey pubK = *getPublicKey*(publicKeyStr);

Signature sig = Signature.*getInstance*(***SIGNATURE\_ALGORITHM***);

sig.initVerify(pubK);

sig.update(data);

**return** sig.verify(sign);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*加密解密\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** **byte**[] encrypt(**byte**[] plainText,String publicKeyStr)**throws** Exception{

PublicKey publicKey = *getPublicKey*(publicKeyStr);

Cipher cipher = Cipher.*getInstance*(***KEY\_ALGORITHM***);

cipher.init(Cipher.***ENCRYPT\_MODE***, publicKey);

**int** inputLen = plainText.length;

ByteArrayOutputStream out = **new** ByteArrayOutputStream();

**int** offSet = 0;

**int** i = 0;

**byte**[] cache;

**while** (inputLen - offSet > 0) {

**if** (inputLen - offSet > ***MAX\_ENCRYPT\_BLOCK***) {

cache = cipher.doFinal(plainText, offSet, ***MAX\_ENCRYPT\_BLOCK***);

} **else** {

cache = cipher.doFinal(plainText, offSet, inputLen - offSet);

}

out.write(cache, 0, cache.length);

i++;

offSet = i \* ***MAX\_ENCRYPT\_BLOCK***;

}

**byte**[] encryptText = out.toByteArray();

out.close();

**return** encryptText;

}

**public** **static** **byte**[] decrypt(**byte**[] encryptText,String privateKeyStr)**throws** Exception{

PrivateKey privateKey = *getPrivateKey*(privateKeyStr);

Cipher cipher = Cipher.*getInstance*(***KEY\_ALGORITHM***);

cipher.init(Cipher.***DECRYPT\_MODE***, privateKey);

**int** inputLen = encryptText.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(encryptText, offSet, ***MAX\_DECRYPT\_BLOCK***);

} **else** {

cache = cipher.doFinal(encryptText, offSet, inputLen - offSet);

}

out.write(cache, 0, cache.length);

i++;

offSet = i \* ***MAX\_DECRYPT\_BLOCK***;

}

**byte**[] plainText = out.toByteArray();

out.close();

**return** plainText;

}

**public** **static** **void** main(String[] args) {

Map<String, RSAKey> 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);

System.***out***.println("测试可行性-------------------");

System.***out***.println("明文======="+input);

cipherText = *encrypt*(input.getBytes(),publicKey);

//加密后的东西

System.***out***.println("Dddddd---------------");

System.***out***.println("密文=======++"+**new** String(cipherText));

System.***out***.println("Dddddd---------------");

//开始解密

**byte**[] plainText = *decrypt*(cipherText,privateKey);

System.***out***.println("解密后明文===== " + **new** String(plainText));

System.***out***.println("验证签名-----------");

String str="被签名的内容";

System.***out***.println("\n原文:"+str);

**byte**[] signature=*sign*(str.getBytes(),privateKey);

**boolean** status=*verify*(str.getBytes(), signature,publicKey);

System.***out***.println("验证情况："+status);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

Idea可以直接使用，但是eclipse会发生错误。解决办法如下：

<https://blog.csdn.net/erlian1992/article/details/79518416>

