<https://blog.csdn.net/luckily01/article/details/7630070>

1. 现在很多J2EE应用都采用一个license文件来授权系统的使用，特别是在系统购买的早期，会提供有限制的license文件对系统进行限制，比如试用版有譬如IP、日期、最大用户数量的限制等。
3. 而license控制的方法又有很多，目前比较流行，只要设计的好就很难破解的方法就是采用一对密匙（私匙加密公匙解密）来生成License文件中的Sinature签名内容，再通过Base64或Hex来进行编码。比如原BEA公司现在是Oracle公司的WebLogic就采用的是这种方法来设置License文件。
5. 这里只进行一个比较简单的实现：
7. 一共三个类：
9. A.KeyGenerater类生成公钥私钥对
11. B.Signaturer类使用私钥进行签名
13. C.SignProvider类用公钥验证
15. 公钥和私钥使用Base64加密Base64这个类很多地方都可以查到。


19. KeyGenerater类：


23. public class KeyGenerater {
25. private byte[] priKey;
27. private byte[] pubKey;
29. public void generater() {
30. try {
32. KeyPairGenerator keygen = KeyPairGenerator .getInstance("RSA");
34. SecureRandom secrand = new SecureRandom();
36. secrand.setSeed("www.川江号子.cn".getBytes()); // 初始化随机产生器
38. keygen.initialize(1024, secrand);
40. KeyPair keys = keygen.genKeyPair();
42. PublicKey pubkey = keys.getPublic();
44. PrivateKey prikey = keys.getPrivate()
46. pubKey = Base64.encodeToByte(pubkey.getEncoded());
48. priKey = Base64.encodeToByte(prikey.getEncoded());
50. System.out.println("pubKey = " + new String(pubKey));
52. System.out.println("priKey = " + new String(priKey));
54. } catch (java.lang.Exception e) {
56. System.out.println("生成密钥对失败");
58. e.printStackTrace();
60. }
62. }
64. public byte[] getPriKey() {
66. return priKey;
68. }
70. public byte[] getPubKey() {
72. return pubKey;
74. }
76. }

79. Signaturer 类：


83. public class Signaturer {
85. public static byte[] sign(byte[] priKeyText, String plainText) {
87. try {
89. PKCS8EncodedKeySpec priPKCS8 = new PKCS8EncodedKeySpec(Base64.decode(priKeyText));
91. KeyFactory keyf = KeyFactory.getInstance("RSA");
93. PrivateKey prikey = keyf.generatePrivate(priPKCS8);
95. // 用私钥对信息生成数字签名
97. Signature signet = java.security.Signature.getInstance("MD5withRSA");
99. signet.initSign(prikey);
101. signet.update(plainText.getBytes());
103. byte[] signed = Base64.encodeToByte(signet.sign());
105. return signed;
107. } catch (java.lang.Exception e) {
109. System.out.println("签名失败");
111. e.printStackTrace();
113. }
115. return null;
117. }
119. }

122. SignProvider 类：
124. public class SignProvider {
126. private SignProvider() {
128. }
130. public static boolean verify(byte[] pubKeyText, String plainText,
132. byte[] signText) {
134. try {
136. // 解密由base64编码的公钥,并构造X509EncodedKeySpec对象
138. X509EncodedKeySpec bobPubKeySpec = new X509EncodedKeySpec(Base64.decode(pubKeyText));
140. // RSA对称加密算法
142. KeyFactory keyFactory = KeyFactory.getInstance("RSA");
144. // 取公钥匙对象
146. PublicKey pubKey = keyFactory.generatePublic(bobPubKeySpec);
148. // 解密由base64编码的数字签名
150. byte[] signed = Base64.decode(signText);
152. Signature signatureChecker = Signature.getInstance("MD5withRSA");
154. signatureChecker.initVerify(pubKey);
156. signatureChecker.update(plainText.getBytes());
158. // 验证签名是否正常
160. if (signatureChecker.verify(signed))
162. return true;
164. else
166. return false;
168. } catch (Throwable e) {
170. System.out.println("校验签名失败");
172. e.printStackTrace();
174. return false;
176. }
178. }
180. }