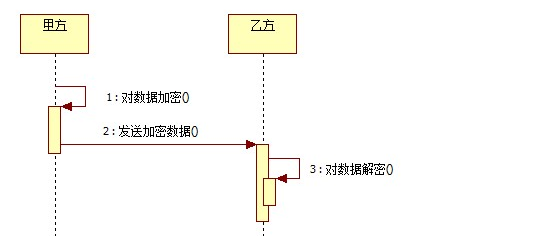
java加密解密算法

参考：http://www.open-open.com/lib/view/open1397274257325.html

# BASE64编码

按 照RFC2045的定义，Base64被定义为：Base64内容传送编码被设计用来把任意序列的8位字节描述为一种不易被人直接识别的形式。（The Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that need not be humanly readable.）

常见于邮件、http加密，截取http信息，你就会发现登录操作的用户名、密码字段通过BASE64加密的。



## 使用jdk实现

|  |
| --- |
| public static void jdkBase64() {  String src = "base64";  try {  //字符串->字节->base64加密后的字符串  BASE64Encoder encoder = new BASE64Encoder();  String encode = encoder.encode(src.getBytes());  System.out.println("encode:" + encode);  //base64字符串解密->字节->字符串  BASE64Decoder decoder = new BASE64Decoder();  System.out.println("decode:" + new String(decoder.decodeBuffer(encode)));  } catch (Exception e) {  e.printStackTrace();  }  }  encode:YmFzZTY0  decode:base64  注意：BASE64Encoder 和BASE64Decoder是非官方JDK实现类。虽然可以在JDK里能找到并使用，但是在API里查不到。JRE 中 sun 和 com.sun 开头包的类都是未被文档化的，他们属于 java, javax 类库的基础，其中的实现大多数与底层平台有关，一般来说是不推荐使用的。 |

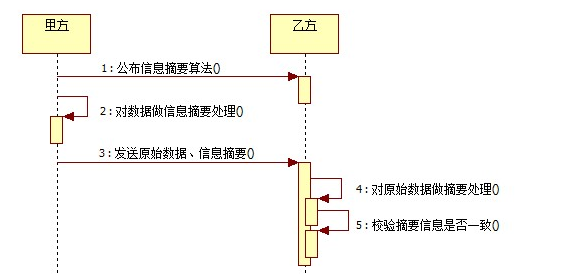
## apache实现

|  |
| --- |
| #maven依赖  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>1.10</version>  </dependency> |
| @Test  public void testApacheBase64() {  String src = "hello world";    byte[] encodeBytes = Base64.encodeBase64(src.getBytes());  System.out.println("common codes encode:" + new String(encodeBytes));  byte[] dencodeBytes = Base64.decodeBase64(encodeBytes);  System.out.println("common codes decode:" + new String(dencodeBytes));  } |

# 信息摘要算法（指纹）

## MD5(Message Digest algorithm 5，信息摘要算法)

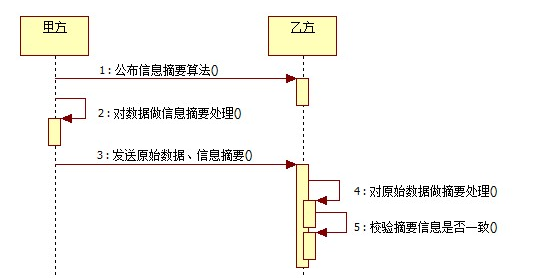
MD5 -- message-digest algorithm 5 （信息-摘要算法）缩写，广泛用于加密和解密技术，常用于文件校验。校验？不管文件多大，经过MD5后都能生成唯一的MD5值。好比现在的ISO校验，都 是MD5校验。怎么用？当然是把ISO经过MD5后产生MD5的值。一般下载linux-ISO的朋友都见过下载链接旁边放着MD5的串。就是用来验证文 件是否一致的。



|  |
| --- |
| @Test  public void jdkTest() {  try {  String src = "hello world";  MessageDigest md = MessageDigest.getInstance("MD5");  byte[] md5Bytes = md.digest(src.getBytes());  System.out.println("JDK MD5:" + Hex.encodeHexString(md5Bytes));  } catch (Exception e) {  e.printStackTrace();  }  }  //JDK MD5:5eb63bbbe01eeed093cb22bb8f5acdc3 |
| @Test  public void CCTest(){  String src = "hello world";  System.out.println("common codes MD5:" + DigestUtils.md5Hex(src.getBytes()));  }  //common codes MD5:5eb63bbbe01eeed093cb22bb8f5acdc3 |

## SHA(Secure Hash Algorithm，安全散列算法)

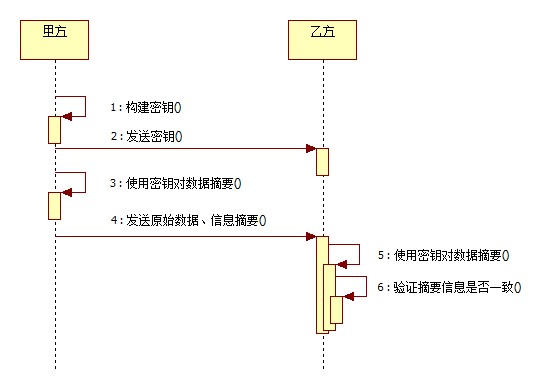
SHA(Secure Hash Algorithm，安全散列算法），数字签名等密码学应用中重要的工具，被广泛地应用于电子商务等信息安全领域。虽然，SHA与MD5通过碰撞法都被破解了， 但是SHA仍然是公认的安全加密算法，较之MD5更为安全。



|  |
| --- |
| @Test  public void jdkTest() {  String src = "hello world";  try {  MessageDigest md = MessageDigest.getInstance("SHA");  md.update(src.getBytes());  System.out.println("jdk sha-1:" + Hex.encodeHexString(md.digest()));  } catch (Exception e) {  e.printStackTrace();  }  }  //jdk sha-1:2aae6c35c94fcfb415dbe95f408b9ce91ee846ed |
| @Test  public void CCTest() {  String src = "hello world";  System.out.println("common codes SHA1 - 1 :" + DigestUtils.sha1Hex(src.getBytes()));  System.out.println("common codes SHA1 - 2 :" + DigestUtils.sha1Hex(src));  System.out.println("common codes SHA512 :" + DigestUtils.sha512Hex(src));  }  common codes SHA1 - 1 :2aae6c35c94fcfb415dbe95f408b9ce91ee846ed  common codes SHA1 - 2 :2aae6c35c94fcfb415dbe95f408b9ce91ee846ed  common codes SHA512 :309ecc489c12d6eb4cc40f50c902f2b4d0ed77ee511a7c7a9bcd3ca86d4cd86f989dd35bc5ff499670da34255b45b0cfd830e81f605dcf7dc5542e93ae9cd76f |

## HMAC(Hash Message Authentication Code散列消息鉴别码)

HMAC(Hash Message Authentication Code，散列消息鉴别码，基于密钥的Hash算法的认证协议。消息鉴别码实现鉴别的原理是，用公开函数和密钥产生一个固定长度的值作为认证标识，用这个 标识鉴别消息的完整性。使用一个密钥生成一个固定大小的小数据块，即MAC，并将其加入到消息中，然后传输。接收方利用与发送方共享的密钥进行鉴别认证 等。



|  |
| --- |
| @Test  public void testHMAC() {  try {  String src = "hello world";  // 初始化KeyGenerator  KeyGenerator keyGenerator = KeyGenerator.getInstance("HmacMD5");  // 产生密钥  SecretKey secretKey = keyGenerator.generateKey();  // 获取密钥  // byte[] key = secretKey.getEncoded();  byte[] key = Hex.decodeHex(new char[] { '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e' });  // 还原密钥  SecretKey restoreSecretKey = new SecretKeySpec(key, "HmacMD5");  // 实例化MAC  Mac mac = Mac.getInstance(restoreSecretKey.getAlgorithm());  // 初始化MAC  mac.init(restoreSecretKey);  // 执行摘要  byte[] hmacMD5Bytes = mac.doFinal(src.getBytes());  System.out.println("jdk hmacMD5:" + Hex.encodeHexString(hmacMD5Bytes));  } catch (Exception e) {  e.printStackTrace();  }  }  //jdk hmacMD5:9d3895d3ebc3ec04b1df79dcdb9a7cff |

# 对称加密算法

## DES(Data Encryption Standard，数据加密算法)

DES-Data Encryption Standard,即数据加密算法。是IBM公司于1975年研究成功并公开发表的。DES算法的入口参数有三个:Key、Data、Mode。其中 Key为8个字节共64位,是DES算法的工作密钥;Data也为8个字节64位,是要被加密或被解密的数据;Mode为DES的工作方式,有两种:加密 或解密。

DES算法把64位的明文输入块变为64位的密文输出块,它所使用的密钥也是64位。

|  |
| --- |
| @Test  public void jdkDES() {  try {  String src = "hello world";  // 生成KEY  KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");  keyGenerator.init(56);  // 产生密钥  SecretKey secretKey = keyGenerator.generateKey();  // 获取密钥  byte[] bytesKey = secretKey.getEncoded();  System.out.println(new String (Base64.encodeBase64(bytesKey)));  // KEY转换  DESKeySpec desKeySpec = new DESKeySpec(bytesKey);  SecretKeyFactory factory = SecretKeyFactory.getInstance("DES");  Key convertSecretKey = factory.generateSecret(desKeySpec);  // 加密  Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");  cipher.init(Cipher.ENCRYPT\_MODE, convertSecretKey);  byte[] result = cipher.doFinal(src.getBytes());  System.out.println("jdk des encrypt:" + Hex.encodeHexString(result));  // 解密  cipher.init(Cipher.DECRYPT\_MODE, convertSecretKey);  result = cipher.doFinal(result);  System.out.println("jdk des decrypt:" + new String(result));  } catch (Exception e) {  e.printStackTrace();  }  }  1ace6b61b30d6157  jdk des encrypt:146d11e25d521069cf0ec48fde09f70b  jdk des decrypt:hello world |

## AES算法

|  |
| --- |
| @Test  public void jdkAES() {  try {  String src = "hello world";  // 生成KEY  KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");  keyGenerator.init(128);  // 产生密钥  SecretKey secretKey = keyGenerator.generateKey();  // 获取密钥  byte[] keyBytes = secretKey.getEncoded();  System.out.println(new String (Base64.encodeBase64(keyBytes)));  // KEY转换  Key key = new SecretKeySpec(keyBytes, "AES");  // 加密  Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");  cipher.init(Cipher.ENCRYPT\_MODE, key);  byte[] result = cipher.doFinal(src.getBytes());  System.out.println("jdk aes encrypt:" + Hex.encodeHexString(result));  // 解密  cipher.init(Cipher.DECRYPT\_MODE, key);  result = cipher.doFinal(result);  System.out.println("jdk aes decrypt:" + new String(result));  } catch (Exception e) {  e.printStackTrace();  }  }  +0tSB0vIqvak+JdV+ZnRgg==  jdk aes encrypt:6eb52a7d8d7421c7829ce273cb963b2c  jdk aes decrypt:hello world |

## PBE(Password-based encryption，基于密码验证)

|  |
| --- |
| @Test  public void PBETest() {  try {  String src = "hello world";  // 初始化盐  SecureRandom random = new SecureRandom();  byte[] salt = random.generateSeed(8);  // 口令与密钥  String password = "timliu";  PBEKeySpec pbeKeySpec = new PBEKeySpec(password.toCharArray());  SecretKeyFactory factory = SecretKeyFactory.getInstance("PBEWITHMD5andDES");  Key key = factory.generateSecret(pbeKeySpec);  // 加密  PBEParameterSpec pbeParameterSpac = new PBEParameterSpec(salt, 100);  Cipher cipher = Cipher.getInstance("PBEWITHMD5andDES");  cipher.init(Cipher.ENCRYPT\_MODE, key, pbeParameterSpac);  byte[] result = cipher.doFinal(src.getBytes());  System.out.println("jdk pbe encrypt:" + Hex.encodeHexString(result));  // 解密  cipher.init(Cipher.DECRYPT\_MODE, key, pbeParameterSpac);  result = cipher.doFinal(result);  System.out.println("jdk pbe decrypt:" + new String(result));  } catch (Exception e) {  e.printStackTrace();  }  }  jdk pbe encrypt:ce0e84d210094a04cf964bcbd2d8183d  jdk pbe decrypt:hello world |

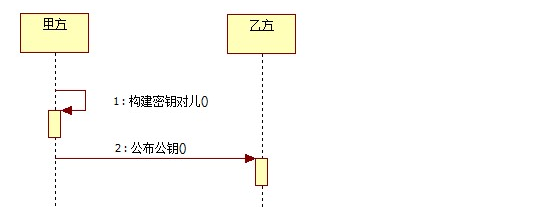
# 非对称加密算法和数字签名

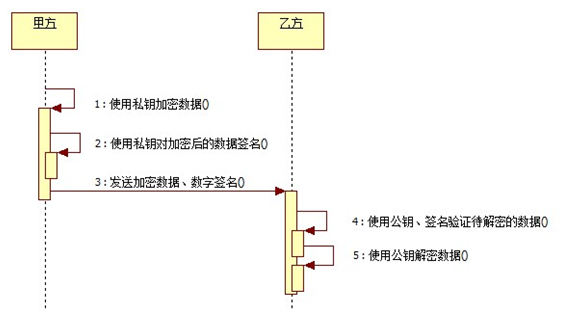
## RSA算法

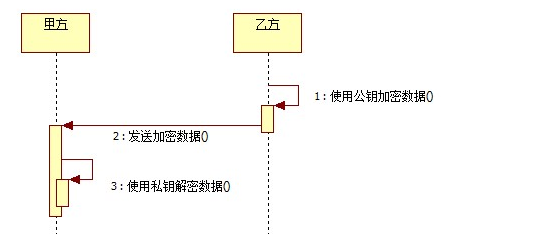
这种算法1978年就出现了，它是第一个既能用于数据加密也能用于数字签名的算法。它易于理解和操作，也很流行。算法的名字以发明者的名字命名：Ron Rivest, AdiShamir 和Leonard Adleman。

这种加密算法的特点主要是密钥的变化，上文我们看到DES只有一个密钥。相当于只有一把钥匙，如果这把钥匙丢了，数据也就不安全了。RSA同时有两把钥 匙，公钥与私钥。同时支持数字签名。数字签名的意义在于，对传输过来的数据进行校验。确保数据在传输工程中不被修改。**流程分析**：

* 甲方构建密钥对儿，将公钥公布给乙方，将私钥保留。
* 甲方使用私钥加密数据，然后用私钥对加密后的数据签名，发送给乙方签名以及加密后的数据；乙方使用公钥、签名来验证待解密数据是否有效，如果有效使用公钥对数据解密。
* 乙方使用公钥加密数据，向甲方发送经过加密后的数据；甲方获得加密数据，通过私钥解密。







|  |
| --- |
| @Test  public void jdkRSA(){  try  {  String src = "hello wprld";  // 1.初始化发送方密钥  KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");  keyPairGenerator.initialize(512);  KeyPair keyPair = keyPairGenerator.generateKeyPair();  RSAPublicKey rsaPublicKey = (RSAPublicKey) keyPair.getPublic();  RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) keyPair.getPrivate();  System.out.println("Public Key:" + Base64.encodeBase64String(rsaPublicKey.getEncoded()));  System.out.println("Private Key:" + Base64.encodeBase64String(rsaPrivateKey.getEncoded()));    // 2.私钥加密、公钥解密 ---- 加密  PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(rsaPrivateKey.getEncoded());  KeyFactory keyFactory = KeyFactory.getInstance("RSA");  PrivateKey privateKey = keyFactory.generatePrivate(pkcs8EncodedKeySpec);  Cipher cipher = Cipher.getInstance("RSA");  cipher.init(Cipher.ENCRYPT\_MODE, privateKey);  byte[] result = cipher.doFinal(src.getBytes());  System.out.println("私钥加密、公钥解密 ---- 加密:" + Base64.encodeBase64String(result));    // 3.私钥加密、公钥解密 ---- 解密  X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(rsaPublicKey.getEncoded());  keyFactory = KeyFactory.getInstance("RSA");  PublicKey publicKey = keyFactory.generatePublic(x509EncodedKeySpec);  cipher = Cipher.getInstance("RSA");  cipher.init(Cipher.DECRYPT\_MODE, publicKey);  result = cipher.doFinal(result);  System.out.println("私钥加密、公钥解密 ---- 解密:" + new String(result));        // 4.公钥加密、私钥解密 ---- 加密  X509EncodedKeySpec x509EncodedKeySpec2 = new X509EncodedKeySpec(rsaPublicKey.getEncoded());  KeyFactory keyFactory2 = KeyFactory.getInstance("RSA");  PublicKey publicKey2 = keyFactory2.generatePublic(x509EncodedKeySpec2);  Cipher cipher2 = Cipher.getInstance("RSA");  cipher2.init(Cipher.ENCRYPT\_MODE, publicKey2);  byte[] result2 = cipher2.doFinal(src.getBytes());  System.out.println("公钥加密、私钥解密 ---- 加密:" + Base64.encodeBase64String(result2));    // 5.私钥解密、公钥加密 ---- 解密  PKCS8EncodedKeySpec pkcs8EncodedKeySpec5 = new PKCS8EncodedKeySpec(rsaPrivateKey.getEncoded());  KeyFactory keyFactory5 = KeyFactory.getInstance("RSA");  PrivateKey privateKey5 = keyFactory5.generatePrivate(pkcs8EncodedKeySpec5);  Cipher cipher5 = Cipher.getInstance("RSA");  cipher5.init(Cipher.DECRYPT\_MODE, privateKey5);  byte[] result5 = cipher5.doFinal(result2);  System.out.println("公钥加密、私钥解密 ---- 解密:" + new String(result5));    } catch (Exception e) {    e.printStackTrace();  }  }  Public Key:MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAJZKL+TCaDBQ30dBgaeEXtoOv3JM+gjKYUP6pAs0yGiN  ovo+Tp7pFguMOX+PFBnLtwE/AkIYE0D725A0eNJbmLsCAwEAAQ==  Private Key:MIIBVAIBADANBgkqhkiG9w0BAQEFAASCAT4wggE6AgEAAkEAlkov5MJoMFDfR0GBp4Re2g6/ckz6  CMphQ/qkCzTIaI2i+j5OnukWC4w5f48UGcu3AT8CQhgTQPvbkDR40luYuwIDAQABAkBdQf3EKupy  CMz0L27f3yDomLwtlhyVNdfzUeVXWtJ9JWhkUHS1n6dlKK/062SdXrfs8XbKd0P7aAc2025oC/KJ  AiEA5CPxIv183CGydMKZT47PRCVziqoHQTzmzEOUymCttQ8CIQCopICem9lwR05gfgOJtGr8z1wS  bVDYJKCkUPHCx8FZlQIgDIAnq1omaE10YM905RYXJ03CCfwND7cNkm21jm1Dt2MCIE2BjTtYFU7s  9rCSOb4IFUOraX1ukEFGVC4wpWwHsm0hAiEA4hM+VHeRp6A/NSqSb8MfJdB4M80g06fvOxk9TYHw  1m4=  私钥加密、公钥解密 ---- 加密:ZF/wRpf6NpR3977EFbIfZFzbYA/pW/J4nNJ3XXymCinBB0WTM1FoDDiPU/S356uih4DtC5mQer6W  Yn7EGKFXbw==  私钥加密、公钥解密 ---- 解密:hello wprld  公钥加密、私钥解密 ---- 加密:BJDUSVk2eXqQh1TOm8Le4MiEt8hC4wN2gbVlwMOO9yAdPcXOdroVb8DpmrEUfCT8FtZOaoJohEWQ  FzA9KlC9xg==  公钥加密、私钥解密 ---- 解密:hello wprld |
| #数字签名  @Test  public void jdkRSASign() {  try {  String src = "hello world";  // 1.初始化密钥  KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");  keyPairGenerator.initialize(512);  KeyPair keyPair = keyPairGenerator.generateKeyPair();  RSAPublicKey rsaPublicKey = (RSAPublicKey) keyPair.getPublic();  RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) keyPair.getPrivate();  // 2.进行签名  PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(rsaPrivateKey.getEncoded());  KeyFactory keyFactory = KeyFactory.getInstance("RSA");  PrivateKey privateKey = keyFactory.generatePrivate(pkcs8EncodedKeySpec);  Signature signature = Signature.getInstance("MD5withRSA");  signature.initSign(privateKey);  signature.update(src.getBytes());  byte[] result = signature.sign();  System.out.println("jdk rsa sign:" + Hex.encodeHexString(result));  // 3.验证签名  X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(rsaPublicKey.getEncoded());  keyFactory = KeyFactory.getInstance("RSA");  PublicKey publicKey = keyFactory.generatePublic(x509EncodedKeySpec);  signature = Signature.getInstance("MD5withRSA");  signature.initVerify(publicKey);  signature.update(src.getBytes());  boolean bool = signature.verify(result);  System.out.println("jdk rsa verify:" + bool);  } catch (Exception e) {  System.out.println(e.toString());  }  }  jdk rsa sign:433eb3ec029f36fc4028879fe96edd436aebd99c6c96e5d89a726847e3f856c4be212378b58c644a890bb9a5e941add028f764ad0b7c9ce885859796e1d2b14b  jdk rsa verify:true |