**[HTTPS那些事 用java实现HTTPS工作原理](http://kingj.iteye.com/blog/2103662)**

[**http://kingj.iteye.com/blog/2103662**](http://kingj.iteye.com/blog/2103662)

**博客分类：**

* [java历险](http://kingj.iteye.com/category/206477)

         今天被问到关于https原理的问题，结果由于知识掌握不牢靠，停留于表面，很多细节都无法回答清楚，于是决定把https的原理弄个明白，废话不多说，我们先看看https的定义

 （由于很久未写博客，排版有些凌乱，请谅解）

**一：什么是https协议**

     在说HTTPS之前先说说什么是HTTP，HTTP就是我们平时浏览网页时候使用的一种协议。HTTP协议传输的数据都是未加密的，也就是明文的，因此使 用HTTP协议传输隐私信息非常不安全。为了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。SSL目前的版本是3.0，被IETF（Internet Engineering Task Force）定义在RFC 6101中，之后IETF对SSL 3.0进行了升级，于是出现了TLS（Transport Layer Security） 1.0，定义在RFC 2246。实际上我们现在的HTTPS都是用的TLS协议，但是由于SSL出现的时间比较早，并且依旧被现在浏览器所支持，因此SSL依然是HTTPS的 代名词，但无论是TLS还是SSL都是上个世纪的事情，SSL最后一个版本是3.0，今后TLS将会继承SSL优良血统继续为我们进行加密服务。目前 TLS的版本是1.2，定义在RFC 5246中，暂时还没有被广泛的使用。对历史感兴趣的朋友可以参考[http://en.wikipedia.org/wiki/Transport\_Layer\_Security](http://en.wikipedia.org/wiki/Transport_Layer_Security" \t "_blank)，这里有对TLS/SSL详尽的叙述。

**二：https的工作原理是什么**

     HTTPS在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握 手过程中将确立双方加密传输数据的密码信息，通常情况下会配合数字证书实现。

TLS/SSL协议不仅仅是一套加密传输的协议，更是一件经过艺术家精心设计的艺术品，TLS/SSL中使用 非对称加密，对称加密以及HASH算法。

这里我们先看看这上面提到的几种技术（如果你对这些技术已经非常了解，那么请跳过该段落，直接到段落三）

1. 数字证书

        数字证书是一种权威性的电子文档,由权威公正的第三方机构，即CA中心签发的证书。它以数字证书为核心的加密技术可以对网络上传输的信息进行加密和解密、数字签名和签名验证，确保网上传递信息的机密性、完整性。 使用了数字证书，即使您发送的信息在网上被他人截获，甚至您丢失了个人的账户、密码等信息，仍可以保证您的账户、资金安全。   
  
     它能提供在Internet上进行身份验证的一种权威性电子文档，人们可以在互联网交往中用它来证明自己的身份和识别对方的身份。当然在数字证书认证的过程中证书认证中心（CA）作为权威的、公正的、可信赖的第三方，其作用是至关重要的.如何判断数字认证中心公正第三方的地位是权威可信的。VeriSign、GeoTrust、Thawte 是国际权威数字证书颁发认证机构的“三巨头”，其中，应用最广的为VerSign签发的电子商务数字证书。  
  
CER(Canonical Encoding Rules,规范编码格式) 是数字证书的一种编码格式，它是BER(Basic Encoding Rules 基本编码格式) 的一个变种，比BER 规定得更严格。后缀的证书文件有两种编码:  
  
DER(Distinguished Encoding Rule 卓越编码格式) 同样是BER的一个变种，DER使用定长模式。  
  
PKCS(Public-Key Cryptography Standards,公钥加密标准) 由RSA实验室和其他安全系统开发商为公钥密码的发展而制定的一系列标准。  
  
pfx是指以pkcs#12格式存储的证书和相应私钥。   
  
在Security编程中，有几种典型的密码交换信息文件格式:   
DER-encoded certificate: .cer, .crt   
PEM-encoded message: .pem   
PKCS#12 Personal Information Exchange: .pfx, .p12   
PKCS#10 Certification Request: .p10 .csr  
PKCS#7 cert request response: .p7r   
PKCS#7 binary message: .p7b .p7c .spc  
  
cer/.crt 是用于存放证书，它是2进制形式存放  
  
pem 跟crt/cer的区别是它以Ascii来表示  
  
pfx/p12 用于存放个人证书/私钥，他通常包含保护密码，2进制方式   
  
p10 .csr 是证书请求   
  
p7r是CA对证书请求的回复，只用于导入   
  
p7b .p7c .spc 以树状展示证书链(certificate chain)，同时也支持单个证书，不含私钥

1. 非对称加密算法

          1976年，美国学者Dime和Henman为解决信息公开传送和密钥管理问题，提出一种新的密钥交换协议，允许在不安全的媒体上的通讯双方交换信息，安全地达成一致的密钥，这就是"公开密钥系统"。相对于"对称加密算法"这种方法也叫做"非对称加密算法"。与对称加密算法不同，非对称加密算法需要两个密钥：公开密钥（publickey）和私有密(privatekey)。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。

       非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将其中的一把作为公用密钥向其它方公开；得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方；甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。甲方只能用其专用密钥解密由其公用密钥加密后的任何信息。非对称加密算法的保密性比较好，它消除了最终用户交换密钥的需要，但加密和解密花费时间长、速度慢，它不适合于对文件加密而只适用于对少量数据进行加密。　经典的非对称加密算法如RSA算法等安全性都相当高. 非对称加密的典型应用是数字签名。采用双钥密码系统的加密方法，在一个过程中使用两个密钥，一个用于加密，另一个用于解密，这种加密方法称为非对称加密，也称为公钥加密，因为其中一个密钥是公开的(另一个则需要保密)。

**DH (Diffie-Hellman)**  
       Diffie-Hellman算法(D-H算法)，密钥一致协议。是由公开密钥密码体制的奠基人Diffie和Hellman所提出的一种思想。简单的说就是允许两名用户在公开媒体上交换信息以生成"一致"的、可以共享的密钥。换句话说，就是由甲方产出一对密钥（公钥、私钥），乙方依照甲方公钥产生乙方密钥对（公钥、私钥）。以此为基线，作为数据传输保密基础，同时双方使用同一种对称加密算法构建本地密钥（SecretKey）对数据加密。这样，在互通了本地密钥（SecretKey）算法后，甲乙双方公开自己的公钥，使用对方的公钥和刚才产生的私钥加密数据，同时可以使用对方的公钥和自己的私钥对数据解密。不单单是甲乙双方两方，可以扩展为多方共享数据通讯，这样就完成了网络交互数据的安全通讯！该算法源于中国的同余定理——中国馀数定理。

**RSA**       RSA公钥加密算法是1977年由Ron Rivest、Adi Shamirh和LenAdleman在（美国麻省理工学院）开发的。RSA取名来自开发他们三者的名字。RSA是目前最有影响力的公钥加密算法，它能够抵抗到目前为止已知的所有密码攻击，已被ISO推荐为公钥数据加密标准。RSA算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但那时想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

**EL Gamal**  
         EL Gamal算法是公钥密码体制中的一种 ,在密码学中占有重要的地位。但该算法所采用的幂剩余计算耗时太多的问题 ,一直是制约其广泛应用的瓶颈问题。提出一种通过建表 ,以及对传统二进制算法进行改进 ,即将指数进行 2 k进制化 ,减少原 BR算法迭代次数 ,提高加密解密速度的算法。

**ECC**  
ECC (Elliptical Curve Cryptography,椭圆曲线加密)算法不椭圆曲线理论为基础，在创建密钥时可以更快，更小，并且更有效，它是用大质数的积来产生。

目前Java 6公提供了DH和RSA两种算法实现，通过Bouncy Castle可以实现Elmal算法支持，另ECC加密算法，目前没有开源组件提支持

1. 对称加密算法

        对加密和解密使用相同密钥的加密算法。由于其速度，对称性加密通常在消息发送方需要加密大量数据时使用。对称性加密也称为密钥加密。对称式数据加密的方式的工作原理如图。所谓对称，就是采用这种加密方法的双方使用方式用同样的密钥进行加密和解密。密钥实际上是一种算法，通信发送方使用这种算法加密数据，接收方再以同样的算法解密数据。因此对称式加密本身不是安全的。常用的对称加密有：

DES、IDEA、RC2、RC4、SKIPJACK算法等 。

采用单钥密码系统的加密方法，同一个密钥可以同时用作信息的加密和解密，这种加密方法称为对称加密，也称为单密钥加密。

1. HASH算法

常用的摘要算法包括MD5，SHA1，SHA256

消息摘要算法的特点：  
  
① 无论输入的消息有多长，计算出来的消息摘要的长度总是固定的。  
② 消息摘要看起来是“随机的”。这些比特看上去是胡乱的杂凑在一起的。  
③ 一般地，只要输入的消息不同，对其进行摘要以后产生的摘要消息也必不相同；但相同的输入必会产生相同的输出。  
④ 消息摘要函数是无陷门的单向函数，即只能进行正向的信息摘要，而无法从摘要中恢复出任何的消息，甚至根本就找不到任何与原信息相关的信息。  
⑤ 好的摘要算法，无法找到两条消息，是它们的摘要相同。

消息摘要（Message Digest）又称为数字摘要(Digital Digest)。它是一个唯一对应一个消息或文本的固定长度的值，它由一个单向Hash加密函数对消息进行作用而产生。如果消息在途中改变了，则接收者通过对收到消息的新产生的摘要与原摘要比较，就可知道消息是否被改变了。因此消息摘要保证了消息的完整性。消息摘要采用单向Hash 函数将需加密 的明文"摘要"成一串128bit的密文，这一串密文亦称为数字指纹(Finger Print)，它有固定的长度，且不同的明文摘要成密文，其结果总是不同的，而同样的明文其摘要必定一致 。这样这串摘要便可成为验证明文是否是"真身"的"指纹"了。

       HASH函数的抗冲突性使得如果一段明文稍有变化，哪怕只更改该段落的一个字母，通过哈希算法作用后都将产生不同的值。而HASH算法的单向性使得要找到到哈希值相同的两个不 同的输入消息，在计算上是不可能的。所以数据的哈希值，即消息摘要，可以检验数据的完整性。哈希函数的这种对不同的输入能够生成不同的值的特性使得无法找到两个具有相同哈希值的输入。因此，如果两个文档经哈希转换后成为相同的值，就可以肯定它们是同一文档。 所以，当希望有效地比较两个数据块时，就可以比较它们的哈希值。例如，可以通过比较邮件发送前和发送后的哈希值来验证该邮件在传递时是否修改

      消息摘要算法的主要特征是加密过程不需要密钥，并且经过加密的数据无法被解密，只有输入相同的明文数据经过相同的消息摘要算法才能得到相同的密文。消息摘要算法不存在 密钥的管理与分发问题，适合于分布式网络相同上使用。由于其加密计算的工作量相当可观，所以以前的这种算法通常只用于数据量有限的情况下的加密，例如计算机的口令就是 用不可逆加密算法加密的。

**三 https握手的过程详细描述**

1.浏览器将自己支持的一套加密规则发送给网站，如RSA加密算法，DES对称加密算法，SHA1摘要算法  
2.网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息（证书中的私钥只能用于服务器端进行解密，在握手的整个过程中，都用到了证书中的公钥和浏览器发送给服务器的随机密码以及对称加密算法）

3.获得网站证书之后浏览器要做以下工作：  
    a) 验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。  
    b) 如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码，并用证书中提供的公钥加密。  
    c) 使用约定好的HASH算法计算握手消息（如SHA1)，并使用生成的随机数对消息进行加密，最后将之前生成的被公钥加密的随机数密码，HASH摘要值一起发送给服务器

4.网站接收浏览器发来的数据之后要做以下的操作：  
    a) 使用自己的私钥将信息解密并取出浏览器发送给服务器的随机密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。  
    b) 使用随机密码加密一段握手消息，发送给浏览器。  
    5.浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密。

从上面的4个大的步骤可以看到，握手的整个过程使用到了数字证书、对称加密、HASH摘要算法，接下来我们用实际代码来实现整个过程

**四 使用java代码模拟整个握手过程**

**一：准备工作**

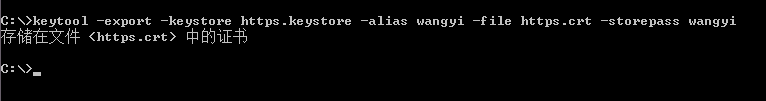
                          1、创建java证书，

                               C:\> keytool -genkey -alias wangyi -keypass wangyi -keyalg RSA -keysize 1024 -keystore https.keystore -storepass wangyi



                           2、将创建的证书保存到C盘（为了方便演示）

                             C:\>keytool -export -keystore https.keystore -alias wangyi -file https.crt -storepass wangyi

**二：代码实现**

                          代码包含6个类，分别为：

|  |  |
| --- | --- |
| 名称 | 说明 |
| CertifcateUtils | 证书操作类 |
| DesCoder | Des对称加密和解密工具类 |
| HttpsMockBase | https父类 |
| HttpsMockClient | client类 |
| HttpsMockServer | 服务器类 |
| SocketUtils | socket工具类 |
| Java代码  [收藏代码](javascript:void())   1. package httpsmock; 3. import java.io.ByteArrayInputStream; 4. import java.io.FileInputStream; 5. import java.io.InputStream; 6. import java.security.KeyStore; 7. import java.security.PrivateKey; 8. import java.security.PublicKey; 9. import java.security.cert.CertificateFactory; 10. /\*\* 11. \* Created by kingj on 2014/8/13. 12. \*/ 13. public class CertifcateUtils { 14. public static byte[] readCertifacates() throws Exception{ 15. CertificateFactory factory=CertificateFactory.getInstance("X.509"); 16. InputStream in=new FileInputStream("c:/https.crt"); 17. java.security.cert.Certificate cate=factory.generateCertificate(in); 18. return cate.getEncoded(); 19. } 21. public static byte[] readPrivateKey() throws  Exception{ 22. KeyStore store=KeyStore.getInstance("JKS"); 23. InputStream in=new FileInputStream("c:/https.keystore"); 24. store.load(in,"wangyi".toCharArray()); 25. PrivateKey pk=(PrivateKey)store.getKey("wangyi","wangyi".toCharArray()); 26. return pk.getEncoded(); 27. } 29. public static PrivateKey readPrivateKeys() throws  Exception{ 30. KeyStore store=KeyStore.getInstance("JKS"); 31. InputStream in=new FileInputStream("c:/https.keystore"); 32. store.load(in,"wangyi".toCharArray()); 33. PrivateKey pk=(PrivateKey)store.getKey("wangyi","wangyi".toCharArray()); 34. return pk; 35. } 37. public static PublicKey readPublicKeys() throws  Exception{ 38. CertificateFactory factory=CertificateFactory.getInstance("X.509"); 39. InputStream in=new FileInputStream("c:/https.crt"); 40. java.security.cert.Certificate cate=factory.generateCertificate(in); 41. return cate.getPublicKey(); 42. } 44. public static  java.security.cert.Certificate createCertiface(byte b[]) throws Exception{ 45. CertificateFactory factory=CertificateFactory.getInstance("X.509"); 46. InputStream in=new ByteArrayInputStream(b); 47. java.security.cert.Certificate cate=factory.generateCertificate(in); 48. return cate; 49. } 51. public static String byte2hex(byte[] b) { 52. String hs = ""; 53. String stmp = ""; 54. for (int n = 0; n < b.length; n++) { 55. stmp = (java.lang.Integer.toHexString(b[n] & 0XFF)); 56. if (stmp.length() == 1) { 57. hs = hs + "0" + stmp; 58. } else { 59. hs = hs + stmp; 60. } 61. } 62. return hs.toUpperCase(); 63. } 64. }     Java代码  [收藏代码](javascript:void())   1. package httpsmock; 3. /\*\* 4. \* Created by kingj on 2014/8/13. 5. \*/ 6. import org.apache.commons.codec.binary.Hex; 8. import java.security.Key; 9. import java.security.SecureRandom; 11. import javax.crypto.Cipher; 12. import javax.crypto.KeyGenerator; 13. import javax.crypto.SecretKey; 14. import javax.crypto.SecretKeyFactory; 15. import javax.crypto.spec.DESKeySpec; 17. /\*\* 18. \* DES Coder<br/> 19. \* secret key length:   56 bit, default:    56 bit<br/> 20. \* mode:    ECB/CBC/PCBC/CTR/CTS/CFB/CFB8 to CFB128/OFB/OBF8 to OFB128<br/> 21. \* padding: Nopadding/PKCS5Padding/ISO10126Padding/ 22. \* @author Aub 23. \* 24. \*/ 25. public class DesCoder { 27. /\*\* 28. \* 密钥算法 29. \*/ 30. private static final String KEY\_ALGORITHM = "DES"; 32. private static final String DEFAULT\_CIPHER\_ALGORITHM = "DES/ECB/PKCS5Padding"; 33. //  private static final String DEFAULT\_CIPHER\_ALGORITHM = "DES/ECB/ISO10126Padding";  36. /\*\* 37. \* 初始化密钥 38. \* 39. \* @return byte[] 密钥 40. \* @throws Exception 41. \*/ 42. public static byte[] initSecretKey(SecureRandom random) throws Exception{ 43. //返回生成指定算法的秘密密钥的 KeyGenerator 对象 44. KeyGenerator kg = KeyGenerator.getInstance(KEY\_ALGORITHM); 45. //初始化此密钥生成器，使其具有确定的密钥大小 46. kg.init(random); 47. //生成一个密钥 48. SecretKey  secretKey = kg.generateKey(); 49. return secretKey.getEncoded(); 50. } 52. /\*\* 53. \* 转换密钥 54. \* 55. \* @param key   二进制密钥 56. \* @return Key  密钥 57. \* @throws Exception 58. \*/ 59. public static Key toKey(byte[] key) throws Exception{ 60. //实例化DES密钥规则 61. DESKeySpec dks = new DESKeySpec(key); 62. //实例化密钥工厂 63. SecretKeyFactory skf = SecretKeyFactory.getInstance(KEY\_ALGORITHM); 64. //生成密钥 65. SecretKey  secretKey = skf.generateSecret(dks); 66. return secretKey; 67. } 69. /\*\* 70. \* 加密 71. \* 72. \* @param data  待加密数据 73. \* @param key   密钥 74. \* @return byte[]   加密数据 75. \* @throws Exception 76. \*/ 77. public static byte[] encrypt(byte[] data,Key key) throws Exception{ 78. return encrypt(data, key,DEFAULT\_CIPHER\_ALGORITHM); 79. } 81. /\*\* 82. \* 加密 83. \* 84. \* @param data  待加密数据 85. \* @param key   二进制密钥 86. \* @return byte[]   加密数据 87. \* @throws Exception 88. \*/ 89. public static byte[] encrypt(byte[] data,byte[] key) throws Exception{ 90. return encrypt(data, key,DEFAULT\_CIPHER\_ALGORITHM); 91. }  94. /\*\* 95. \* 加密 96. \* 97. \* @param data  待加密数据 98. \* @param key   二进制密钥 99. \* @param cipherAlgorithm   加密算法/工作模式/填充方式 100. \* @return byte[]   加密数据 101. \* @throws Exception 102. \*/ 103. public static byte[] encrypt(byte[] data,byte[] key,String cipherAlgorithm) throws Exception{ 104. //还原密钥 105. Key k = toKey(key); 106. return encrypt(data, k, cipherAlgorithm); 107. } 109. /\*\* 110. \* 加密 111. \* 112. \* @param data  待加密数据 113. \* @param key   密钥 114. \* @param cipherAlgorithm   加密算法/工作模式/填充方式 115. \* @return byte[]   加密数据 116. \* @throws Exception 117. \*/ 118. public static byte[] encrypt(byte[] data,Key key,String cipherAlgorithm) throws Exception{ 119. //实例化 120. Cipher cipher = Cipher.getInstance(cipherAlgorithm); 121. //使用密钥初始化，设置为加密模式 122. cipher.init(Cipher.ENCRYPT\_MODE, key); 123. //执行操作 124. return cipher.doFinal(data); 125. }   129. /\*\* 130. \* 解密 131. \* 132. \* @param data  待解密数据 133. \* @param key   二进制密钥 134. \* @return byte[]   解密数据 135. \* @throws Exception 136. \*/ 137. public static byte[] decrypt(byte[] data,byte[] key) throws Exception{ 138. return decrypt(data, key,DEFAULT\_CIPHER\_ALGORITHM); 139. } 141. /\*\* 142. \* 解密 143. \* 144. \* @param data  待解密数据 145. \* @param key   密钥 146. \* @return byte[]   解密数据 147. \* @throws Exception 148. \*/ 149. public static byte[] decrypt(byte[] data,Key key) throws Exception{ 150. return decrypt(data, key,DEFAULT\_CIPHER\_ALGORITHM); 151. } 153. /\*\* 154. \* 解密 155. \* 156. \* @param data  待解密数据 157. \* @param key   二进制密钥 158. \* @param cipherAlgorithm   加密算法/工作模式/填充方式 159. \* @return byte[]   解密数据 160. \* @throws Exception 161. \*/ 162. public static byte[] decrypt(byte[] data,byte[] key,String cipherAlgorithm) throws Exception{ 163. //还原密钥 164. Key k = toKey(key); 165. return decrypt(data, k, cipherAlgorithm); 166. } 168. /\*\* 169. \* 解密 170. \* 171. \* @param data  待解密数据 172. \* @param key   密钥 173. \* @param cipherAlgorithm   加密算法/工作模式/填充方式 174. \* @return byte[]   解密数据 175. \* @throws Exception 176. \*/ 177. public static byte[] decrypt(byte[] data,Key key,String cipherAlgorithm) throws Exception{ 178. //实例化 179. Cipher cipher = Cipher.getInstance(cipherAlgorithm); 180. //使用密钥初始化，设置为解密模式 181. cipher.init(Cipher.DECRYPT\_MODE, key); 182. //执行操作 183. return cipher.doFinal(data); 184. } 186. private static String  showByteArray(byte[] data){ 187. if(null == data){ 188. return null; 189. } 190. StringBuilder sb = new StringBuilder("{"); 191. for(byte b:data){ 192. sb.append(b).append(","); 193. } 194. sb.deleteCharAt(sb.length()-1); 195. sb.append("}"); 196. return sb.toString(); 197. } 199. }       Java代码  [收藏代码](javascript:void())   1. package httpsmock; 3. import com.sun.org.apache.bcel.internal.generic.NEW; 5. import javax.crypto.\*; 6. import javax.crypto.spec.DESKeySpec; 7. import java.security.\*; 8. import java.security.spec.InvalidKeySpecException; 9. import java.util.Random; 11. /\*\* 12. \* Created by kingj on 2014/8/13. 13. \*/ 14. public class HttpsMockBase { 15. static PrivateKey privateKey; 16. static PublicKey publicKey;  19. public static boolean byteEquals(byte a[],byte[] b){ 20. boolean equals=true; 21. if(a==null || b==null){ 22. equals=false; 23. } 25. if(a!=null && b!=null){ 26. if(a.length!=b.length){ 27. equals=false; 28. }else{ 29. for(int i=0;i<a.length;i++){ 30. if(a[i]!=b[i]){ 31. equals=false; 32. break; 33. } 34. } 35. } 37. } 38. return equals; 39. } 41. public static byte[] decrypt(byte data[]) throws Exception{ 42. // 对数据解密 43. Cipher cipher = Cipher.getInstance(privateKey.getAlgorithm()); 44. cipher.init(Cipher.DECRYPT\_MODE, privateKey); 45. return cipher.doFinal(data); 46. } 48. public static byte[] decrypt(byte data[],SecureRandom seed) throws Exception{ 49. // 对数据解密 50. Cipher cipher = Cipher.getInstance(privateKey.getAlgorithm()); 51. cipher.init(Cipher.DECRYPT\_MODE, privateKey,seed); 52. return cipher.doFinal(data); 53. } 55. public static byte[] decryptByPublicKey(byte data[],SecureRandom seed) throws Exception{ 56. if(publicKey==null){ 57. publicKey=CertifcateUtils.readPublicKeys(); 58. } 59. // 对数据解密 60. Cipher cipher = Cipher.getInstance(publicKey.getAlgorithm()); 61. if(seed==null){ 62. cipher.init(Cipher.DECRYPT\_MODE, publicKey); 63. }else{ 64. cipher.init(Cipher.DECRYPT\_MODE, publicKey,seed); 65. } 67. return cipher.doFinal(data); 68. } 70. public static byte[] decryptByDes(byte data[],SecureRandom seed) throws Exception{ 71. if(publicKey==null){ 72. publicKey=CertifcateUtils.readPublicKeys(); 73. } 74. // 对数据解密 75. Cipher cipher = Cipher.getInstance("DES"); 76. if(seed==null){ 77. cipher.init(Cipher.DECRYPT\_MODE, publicKey); 78. }else{ 79. cipher.init(Cipher.DECRYPT\_MODE, publicKey,seed); 80. } 82. return cipher.doFinal(data); 83. }    88. public static byte[] encryptByPublicKey(byte[] data, SecureRandom seed) 89. throws Exception { 90. if(publicKey==null){ 91. publicKey=CertifcateUtils.readPublicKeys(); 92. } 93. // 对数据加密 94. Cipher cipher = Cipher.getInstance(publicKey.getAlgorithm()); 95. if(seed==null){ 96. cipher.init(Cipher.ENCRYPT\_MODE, publicKey); 97. }else{ 98. cipher.init(Cipher.ENCRYPT\_MODE, publicKey,seed); 99. } 101. return cipher.doFinal(data); 102. } 104. public static String byte2hex(byte[] b) { 105. String hs = ""; 106. String stmp = ""; 107. for (int n = 0; n < b.length; n++) { 108. stmp = (Integer.toHexString(b[n] & 0XFF)); 109. if (stmp.length() == 1) { 110. hs = hs + "0" + stmp; 111. } else { 112. hs = hs +"  " + stmp; 113. } 114. } 115. return hs.toUpperCase(); 116. } 118. public static byte[] cactHash(byte[] bytes) { 119. byte[] \_bytes = null; 120. try { 121. MessageDigest md = MessageDigest.getInstance("SHA1"); 122. md.update(bytes); 123. \_bytes = md.digest(); 124. } catch (NoSuchAlgorithmException ex) { 125. ex.printStackTrace(); 126. } 127. return \_bytes; 128. }   132. static String random(){ 133. StringBuilder builder=new StringBuilder(); 134. Random random=new Random(); 135. int seedLength=10; 136. for(int i=0;i<seedLength;i++){ 137. builder.append(digits[random.nextInt(seedLength)]); 138. } 140. return builder.toString(); 141. } 143. static char[] digits={ 144. '0','1','2','3','4', 145. '5','6','7','8','9', 146. 'a','b','c','d','e', 147. 'f','g','h','i','j' 148. }; 150. }     Java代码  [收藏代码](javascript:void())   1. package httpsmock; 3. import java.io.DataInputStream; 4. import java.io.DataOutputStream; 5. import java.net.Socket; 6. import java.security.Key; 7. import java.security.SecureRandom; 9. /\*\* 10. \* Created by kingj on 2014/8/13. 11. \*/ 12. public class HttpsMockClient extends  HttpsMockBase { 13. static DataInputStream in; 14. static DataOutputStream out; 15. static Key key; 16. public static void main(String args[]) throws  Exception{ 17. int port=80; 18. Socket s=new Socket("localhost",port); 19. s.setReceiveBufferSize(102400); 20. s.setKeepAlive(true); 21. in=new DataInputStream(s.getInputStream()); 22. out=new DataOutputStream(s.getOutputStream()); 23. shakeHands(); 25. System.out.println("------------------------------------------------------------------"); 26. String name="duck"; 27. writeBytes(name.getBytes()); 29. int len=in.readInt(); 30. byte[] msg=readBytes(len); 31. System.out.println("服务器反馈消息:"+byte2hex(msg)); 32. Thread.sleep(1000\*100);  35. } 37. private static void shakeHands() throws Exception { 38. //第一步 客户端发送自己支持的hash算法 39. String supportHash="SHA1"; 40. int length=supportHash.getBytes().length; 41. out.writeInt(length); 42. SocketUtils.writeBytes(out, supportHash.getBytes(), length); 44. //第二步 客户端验证服务器端证书是否合法 45. int skip=in.readInt(); 46. byte[] certificate=SocketUtils.readBytes(in,skip); 47. java.security.cert.Certificate cc= CertifcateUtils.createCertiface(certificate); 49. publicKey=cc.getPublicKey(); 50. cc.verify(publicKey); 51. System.out.println("客户端校验服务器端证书是否合法：" +true); 53. //第三步  客户端校验服务器端发送过来的证书成功,生成随机数并用公钥加密 54. System.out.println("客户端校验服务器端发送过来的证书成功,生成随机数并用公钥加密"); 55. SecureRandom seed=new SecureRandom(); 56. int seedLength=2; 57. byte seedBytes[]=seed.generateSeed(seedLength); 58. System.out.println("生成的随机数为 : " + byte2hex(seedBytes)); 59. System.out.println("将随机数用公钥加密后发送到服务器"); 60. byte[] encrptedSeed=encryptByPublicKey(seedBytes, null); 61. SocketUtils.writeBytes(out,encrptedSeed,encrptedSeed.length); 63. System.out.println("加密后的seed值为 :" + byte2hex(encrptedSeed)); 65. String message=random(); 66. System.out.println("客户端生成消息为:"+message); 68. System.out.println("使用随机数并用公钥对消息加密"); 69. byte[] encrpt=encryptByPublicKey(message.getBytes(),seed); 70. System.out.println("加密后消息位数为 : " +encrpt.length); 71. SocketUtils.writeBytes(out,encrpt,encrpt.length); 73. System.out.println("客户端使用SHA1计算消息摘要"); 74. byte hash[]=cactHash(message.getBytes()); 75. System.out.println("摘要信息为:"+byte2hex(hash)); 77. System.out.println("消息加密完成，摘要计算完成，发送服务器"); 78. SocketUtils.writeBytes(out,hash,hash.length);  81. System.out.println("客户端向服务器发送消息完成，开始接受服务器端发送回来的消息和摘要"); 82. System.out.println("接受服务器端发送的消息"); 83. int serverMessageLength=in.readInt(); 84. byte[] serverMessage=SocketUtils.readBytes(in,serverMessageLength); 85. System.out.println("服务器端的消息内容为 ：" + byte2hex(serverMessage)); 87. System.out.println("开始用之前生成的随机密码和DES算法解密消息,密码为:"+byte2hex(seedBytes)); 88. byte[] desKey= DesCoder.initSecretKey(new SecureRandom(seedBytes)); 89. key=DesCoder.toKey(desKey); 91. byte[] decrpytedServerMsg=DesCoder.decrypt(serverMessage, key); 92. System.out.println("解密后的消息为:"+byte2hex(decrpytedServerMsg)); 94. int serverHashLength=in.readInt(); 95. byte[] serverHash=SocketUtils.readBytes(in,serverHashLength); 96. System.out.println("开始接受服务器端的摘要消息:"+byte2hex(serverHash)); 98. byte[] serverHashValues=cactHash(decrpytedServerMsg); 99. System.out.println("计算服务器端发送过来的消息的摘要 : " +byte2hex(serverHashValues)); 101. System.out.println("判断服务器端发送过来的hash摘要是否和计算出的摘要一致"); 102. boolean isHashEquals=byteEquals(serverHashValues,serverHash); 104. if(isHashEquals){ 105. System.out.println("验证完成，握手成功"); 106. }else{ 107. System.out.println("验证失败，握手失败"); 108. } 109. }  112. public static byte[] readBytes(int length) throws  Exception{ 113. byte[] undecrpty=SocketUtils.readBytes(in,length); 114. System.out.println("读取未解密消息:"+byte2hex(undecrpty)); 115. return DesCoder.decrypt(undecrpty,key); 116. } 118. public static void writeBytes(byte[] data) throws  Exception{ 119. byte[] encrpted=DesCoder.encrypt(data,key); 120. System.out.println("写入加密后消息:"+byte2hex(encrpted)); 121. SocketUtils.writeBytes(out,encrpted,encrpted.length); 122. } 123. }     Java代码  [收藏代码](javascript:void())   1. package httpsmock; 3. import javax.net.ServerSocketFactory; 4. import java.io.DataInputStream; 5. import java.io.DataOutputStream; 6. import java.net.ServerSocket; 7. import java.net.Socket; 8. import java.security.Key; 9. import java.security.SecureRandom; 10. import java.util.concurrent.ExecutorService; 11. import java.util.concurrent.Executors; 13. /\*\* 14. \* Created by kingj on 2014/8/13. 15. \*/ 16. public class HttpsMockServer extends HttpsMockBase { 17. static DataInputStream in; 18. static DataOutputStream out; 19. static String hash; 20. static Key key; 21. static ExecutorService executorService= Executors.newFixedThreadPool(20); 22. public static void main(String args[]) throws Exception{ 23. int port=80; 24. ServerSocket ss= ServerSocketFactory.getDefault().createServerSocket(port); 25. ss.setReceiveBufferSize(102400); 26. ss.setReuseAddress(false); 27. while(true){ 28. try { 29. final Socket s = ss.accept(); 30. doHttpsShakeHands(s); 31. executorService.execute(new Runnable() { 32. @Override 33. public void run() { 34. doSocketTransport(s); 35. } 36. }); 38. }catch (Exception e){ 39. e.printStackTrace(); 40. } 41. } 42. } 44. private static void doSocketTransport(Socket s){ 45. try{ 46. System.out.println("--------------------------------------------------------"); 47. int length=in.readInt(); 48. byte[] clientMsg=readBytes(length); 49. System.out.println("客户端指令内容为:" + byte2hex(clientMsg)); 51. writeBytes("服务器已经接受请求".getBytes()); 52. }catch (Exception ex){ 53. ex.printStackTrace(); 54. } 55. } 57. public static byte[] readBytes(int length) throws  Exception{ 58. byte[] undecrpty=SocketUtils.readBytes(in,length); 59. System.out.println("读取未解密消息:"+byte2hex(undecrpty)); 60. return DesCoder.decrypt(undecrpty,key); 61. } 63. public static void writeBytes(byte[] data) throws  Exception{ 64. byte[] encrpted=DesCoder.encrypt(data,key); 65. System.out.println("写入加密后消息:"+byte2hex(encrpted)); 66. SocketUtils.writeBytes(out,encrpted,encrpted.length); 67. } 69. private static void doHttpsShakeHands(Socket s) throws Exception { 70. in=new DataInputStream(s.getInputStream()); 71. out=new DataOutputStream(s.getOutputStream()); 73. //第一步 获取客户端发送的支持的验证规则，包括hash算法，这里选用SHA1作为hash 74. int length=in.readInt(); 75. in.skipBytes(4); 76. byte[] clientSupportHash=SocketUtils.readBytes(in,length); 77. String clientHash=new String(clientSupportHash); 78. hash=clientHash; 79. System.out.println("客户端发送了hash算法为:"+clientHash); 81. //第二步，发送服务器证书到客户端 82. byte[] certificateBytes=CertifcateUtils.readCertifacates(); 83. privateKey=CertifcateUtils.readPrivateKeys(); 84. System.out.println("发送证书给客户端,字节长度为:"+certificateBytes.length); 85. System.out.println("证书内容为:" + byte2hex(certificateBytes)); 86. SocketUtils.writeBytes(out, certificateBytes, certificateBytes.length); 88. System.out.println("获取客户端通过公钥加密后的随机数"); 89. int secureByteLength=in.readInt(); 90. byte[] secureBytes=SocketUtils.readBytes(in, secureByteLength); 92. System.out.println("读取到的客户端的随机数为:"+byte2hex(secureBytes)); 93. byte secureSeed[]=decrypt(secureBytes); 94. System.out.println("解密后的随机数密码为:" +byte2hex(secureSeed)); 96. //第三步 获取客户端加密字符串 97. int skip=in.readInt(); 98. System.out.println("第三步 获取客户端加密消息,消息长度为 ：" +skip); 99. byte[] data=SocketUtils.readBytes(in,skip); 101. System.out.println("客户端发送的加密消息为 : " +byte2hex(data)); 102. System.out.println("用私钥对消息解密，并计算SHA1的hash值"); 103. byte message[] =decrypt(data,new SecureRandom(secureBytes)); 104. byte serverHash[]=cactHash(message);  107. System.out.println("获取客户端计算的SHA1摘要"); 108. int hashSkip=in.readInt(); 109. byte[] clientHashBytes=SocketUtils.readBytes(in,hashSkip); 110. System.out.println("客户端SHA1摘要为 : " + byte2hex(clientHashBytes)); 112. System.out.println("开始比较客户端hash和服务器端从消息中计算的hash值是否一致"); 113. boolean isHashEquals=byteEquals(serverHash,clientHashBytes); 114. System.out.println("是否一致结果为 ： " + isHashEquals);   118. System.out.println("第一次校验客户端发送过来的消息和摘译一致，服务器开始向客户端发送消息和摘要"); 119. System.out.println("生成密码用于加密服务器端消息,secureRandom : "+byte2hex(secureSeed)); 120. SecureRandom secureRandom=new SecureRandom(secureSeed); 122. String randomMessage=random(); 123. System.out.println("服务器端生成的随机消息为 : "+randomMessage); 125. System.out.println("用DES算法并使用客户端生成的随机密码对消息加密"); 126. byte[] desKey=DesCoder.initSecretKey(secureRandom); 127. key=DesCoder.toKey(desKey); 129. byte serverMessage[]=DesCoder.encrypt(randomMessage.getBytes(), key); 130. SocketUtils.writeBytes(out,serverMessage,serverMessage.length); 131. System.out.println("服务器端发送的机密后的消息为:"+byte2hex(serverMessage)+",加密密码为:"+byte2hex(secureSeed)); 133. System.out.println("服务器端开始计算hash摘要值"); 134. byte serverMessageHash[]=cactHash(randomMessage.getBytes()); 135. System.out.println("服务器端计算的hash摘要值为 :" +byte2hex(serverMessageHash)); 136. SocketUtils.writeBytes(out,serverMessageHash,serverMessageHash.length); 138. System.out.println("握手成功，之后所有通信都将使用DES加密算法进行加密"); 139. } 141. }     Java代码  [收藏代码](javascript:void())   1. package httpsmock; 3. import java.io.ByteArrayInputStream; 4. import java.io.DataInputStream; 5. import java.io.DataOutputStream; 6. import java.io.IOException; 7. import java.net.Socket; 8. import java.util.Arrays; 10. /\*\* 11. \* Created by kingj on 2014/8/13. 12. \*/ 13. public class SocketUtils { 14. public static void close(Socket s){ 15. try { 16. s.shutdownInput(); 17. s.shutdownOutput(); 18. } catch (IOException e) { 19. e.printStackTrace(); 20. } 21. }   25. public static byte[] readBytes(DataInputStream in,int length) throws IOException { 26. int r=0; 27. byte[] data=new byte[length]; 28. while(r<length){ 29. r+=in.read(data,r,length-r); 30. } 32. return data; 33. } 35. public static void writeBytes(DataOutputStream out,byte[] bytes,int length) throws IOException{ 36. out.writeInt(length); 37. out.write(bytes,0,length); 38. out.flush(); 39. } 40. }           通过运行上述代码，我们可以看看服务器端和客户端控制台打印的消息记录（https握手完成后，整个过程数据传输都需要客户端和服务端使用约定的DES算法对数据进行加密和解密）      1、服务端消息记录  客户端发送了hash算法为:SHA1 发送证书给客户端,字节长度为:618 证书内容为:  30  8202  66  30  8201  CF  A0030201020204  51  84  FA  AF  300D0609  2A  86  48  86  F70D01010B0500  30  66  310F  300D0603  550406  1306  77  61  6E  67  79  69  310F  300D0603  550408  1306  77  61  6E  67  79  69  310F  300D0603  550407  1306  77  61  6E  67  79  69  310F  300D0603  55040A  1306  77  61  6E  67  79  69  310F  300D0603  55040B  1306  77  61  6E  67  79  69  310F  300D0603  550403  1306  77  61  6E  67  79  69  30  1E  170D  31  34  30  38  31  33  30  35  32  30  35  34  5A  170D  31  34  31  31  31  31  30  35  32  30  35  34  5A  30  66  310F  300D0603  550406  1306  77  61  6E  67  79  69  310F  300D0603  550408  1306  77  61  6E  67  79  69  310F  300D0603  550407  1306  77  61  6E  67  79  69  310F  300D0603  55040A  1306  77  61  6E  67  79  69  310F  300D0603  55040B  1306  77  61  6E  67  79  69  310F  300D0603  550403  1306  77  61  6E  67  79  69  30  81  9F  300D0609  2A  86  48  86  F70D010101050003  81  8D00  30  81  8902  81  8100  89  20  2A  F6  BF  1E  F9  95  F8  E5  E2  C2  C6  14  22  DB  23  10  2F  44  E0  AD0B  FB  89  62  8C  A6  E2  14  52  E7  5D  FE  7B  CC  A4  D2  F4  F9  C5  8E  E0  75  CC  F3  71  E9  29  85  A9  DA  D2  BD  93  73  12  74  2B  4C  D2  74  1A  13  82  64  20  E0  8B  68  FF  9A  F0  6F0C  880F  91  A5  FE  42  44  DE  81  F0  47  C7  67  2001  C7  7E  8B  36  87  E8  1B  7E  6907  D0  39  77  DE  53  D4  F5  67  57  BD  15  8E  51  E5  44  10  CD  BE  81  EB  E3  86  E8  73  B5  1D  1F  FF0203010001  A3  21  30  1F  30  1D0603  55  1D0E04  1604  14  E2  81  F2  3E  81  92  8B  DE  7A  1D  93  A9  28  23  A7  5D  E7  65  63  EB  300D0609  2A  86  48  86  F70D01010B050003  81  810002  E6  BF00  FB  CE  3A  4A  AC  9E  5F  10  6C  4F  FE  44  93  A4  6D  89  BC  4F  CB  25  30  1F  B4  C7  67  E3  E6  A1  1D  66  4B  DA  E4  6D  D8  90  CC  D2  74  34  48  6C  9B  33  2E  C2  4E  9E  AA  470B  9B  4000  7A  59  67  3E  C2  75  1A  A0  7A  48  16  53  D6  C4  53  97080B  F4  23  49  2E06  60  DF  9D  B4  5B  76  B2  AC  35  CF  2E  3C  CA  E3  B6  25  7D  F7  BA  69  6F  15  CE  AF  B4  9D  83  94  2E  5E  37  6E  C5  C2  B9  94  54  DB06  5D  7F  B6  70  1C  91  E6  E3 获取客户端通过公钥加密后的随机数 读取到的客户端的随机数为:  86  16  A9  65  F6  EC  A3  57  D6  23  A2  43  8F  F4  52  F5  37  14  F9  5B  27  6F  75  A3  25  C9  9E  D4  DD  CC  68  BA03  A2  1B  E6  8D  74  61  3B  28  28  9F  1F  5A  AD  5F  32  4B  40  81  98  54  AC0F  840B  80  BF  53  80  50  1E  A7  24  16  10  2A  2B  6A  8709  86  7C  20  75  20  14  7E  38  F3  FA  76  6207  D1  E1  37  28  93  D9  C1  2F  D4  9B  6E  9A  5205  9A  6D  54  8B  DD  1D  8205  DF  BC  AE  BB  6C  24  F5  6E  BC  F2  DE  26  AB  B1  87  1F  DA  DE  3B  25  1E 解密后的随机数密码为:  5B  D4 第三步 获取客户端加密消息,消息长度为 ：128 客户端发送的加密消息为 :   32  76  EB  3E  93  E7  F1  590E  67  EB  FA  29  24  5D  F4  A2  3E  78  BE  61  49  B1  4C  91  1A  450A  B7  D7  E0  71  A1  30  C0  12  F905  9C  CF  B9  C9  75  6B  C4  39  3C  EF  5F  1005  75  AD  50  9A09  6F  8A  7F  C0  F4  20  E0  BC  DF  74  90  F3  6A  46  5E  6C  47  FC  16  EC  4D  DD  10  F9  87  ED  E4  47  83  37  B8  6A  5B  5B  B2  17  9306  7707  72  8E  3008  73  59  89  F5  F7  E6  66  89  4F  F7  B6  2B  41  7B  3B  1B  29  63  D0  11  D4  52  60  4A  3B  74  CA  1E 用私钥对消息解密，并计算SHA1的hash值 获取客户端计算的SHA1摘要 客户端SHA1摘要为 : 01  56  CB  DF  D3  EF  5A  8F  BB  85  BE  15  FB  83  D9  10  1F  64  F6  D8 开始比较客户端hash和服务器端从消息中计算的hash值是否一致 是否一致结果为 ： true 第一次校验客户端发送过来的消息和摘译一致，服务器开始向客户端发送消息和摘要 生成密码用于加密服务器端消息,secureRandom :   5B  D4  （使用客户端第一次传过来的密码） 服务器端生成的随机消息为 : 2355384499 用DES算法并使用客户端生成的随机密码对消息加密 服务器端发送的机密后的消息为:  34  DE  39  CE  7A  280D  4F  44  83  51  2D  C3  EB  4F  1B,加密密码为:  5B  D4   （使用客户端第一次传过来的密码） 服务器端开始计算hash摘要值 服务器端计算的hash摘要值为 :  DD  3D  66  B5  C8  B6  A2  36  5E  D1  55  9A  B6  F7  C0  39  3C  97  1402 握手成功，之后所有通信都将使用DES加密算法进行加密 -------------------------------------------------------- 读取未解密消息:  9D  2D  C2  D7  5D  2F  3C  F5 客户端指令内容为:  64  75  63  6B 写入加密后消息:  52  91  2C  62  E3  B9  5E  80  CF  3D  39  B4  7D  55  B7  3A  97  46  34  98  5603  DA  FC  A9  E1  D1  61  8F  24  64  D8        2、客户端消息记录    客户端校验服务器端证书是否合法：true (校验证书) 客户端校验服务器端发送过来的证书成功,生成随机数并用公钥加密 生成的随机数为 :   5B  D4  （客户端生成了随机密码，用于整个握手过程中） 将随机数用公钥加密后发送到服务器 加密后的seed值为 :  86  16  A9  65  F6  EC  A3  57  D6  23  A2  43  8F  F4  52  F5  37  14  F9  5B  27  6F  75  A3  25  C9  9E  D4  DD  CC  68  BA03  A2  1B  E6  8D  74  61  3B  28  28  9F  1F  5A  AD  5F  32  4B  40  81  98  54  AC0F  840B  80  BF  53  80  50  1E  A7  24  16  10  2A  2B  6A  8709  86  7C  20  75  20  14  7E  38  F3  FA  76  6207  D1  E1  37  28  93  D9  C1  2F  D4  9B  6E  9A  5205  9A  6D  54  8B  DD  1D  8205  DF  BC  AE  BB  6C  24  F5  6E  BC  F2  DE  26  AB  B1  87  1F  DA  DE  3B  25  1E 客户端生成消息为:9080292229 使用随机数并用公钥对消息加密 加密后消息位数为 : 128 客户端使用SHA1计算消息摘要 摘要信息为:01  56  CB  DF  D3  EF  5A  8F  BB  85  BE  15  FB  83  D9  10  1F  64  F6  D8 消息加密完成，摘要计算完成，发送服务器 客户端向服务器发送消息完成，开始接受服务器端发送回来的消息和摘要 接受服务器端发送的消息 服务器端的消息内容为 ：  34  DE  39  CE  7A  280D  4F  44  83  51  2D  C3  EB  4F  1B 开始用之前生成的随机密码和DES算法解密消息,密码为:  5B  D4 解密后的消息为:  32  33  35  35  33  38  34  34  39  39 开始接受服务器端的摘要消息:  DD  3D  66  B5  C8  B6  A2  36  5E  D1  55  9A  B6  F7  C0  39  3C  97  1402 计算服务器端发送过来的消息的摘要 :   DD  3D  66  B5  C8  B6  A2  36  5E  D1  55  9A  B6  F7  C0  39  3C  97  1402 判断服务器端发送过来的hash摘要是否和计算出的摘要一致 验证完成，握手成功 ------------------------------------------------------------------ 写入加密后消息:  9D  2D  C2  D7  5D  2F  3C  F5 读取未解密消息:  52  91  2C  62  E3  B9  5E  80  CF  3D  39  B4  7D  55  B7  3A  97  46  34  98  5603  DA  FC  A9  E1  D1  61  8F  24  64  D8 服务器反馈消息:  E6  9C  8D  E5  8A  A1  E5  99  A8  E5  B7  B2  E7  BB  8F  E6  8E  A5  E5  8F  97  E8  AF  B7  E6  B1  82 |  |