SSL通信过程分析

一、SSL建立握手连接目的

1.身份的验证，client与server确认对方是它相连接的，而不是第三方冒充的，通过证书实现。

2.client与server交换session key，用于连接后数据的传输加密和hash校验。

二、简单的SSL握手连接过程

(仅Server端交换证书给client)：

1.client发送ClientHello，指定版本，随机数(RN)，所有支持的密码套件(CipherSuites)

2.server回应ServerHello，指定版本，RN，选择CipherSuites，会话ID(Session ID)

3.server发送Certificate

4.Server发送ServerHelloDone

5.Client发送ClientKeyExchange，用于与server交换session key

6.Client发送ChangeCipherSpec，指示Server从现在开始发送的消息都是加密过的

7.Client发送Finishd，包含了前面所有握手消息的hash，可以让server验证握手过程是否被第三方篡改

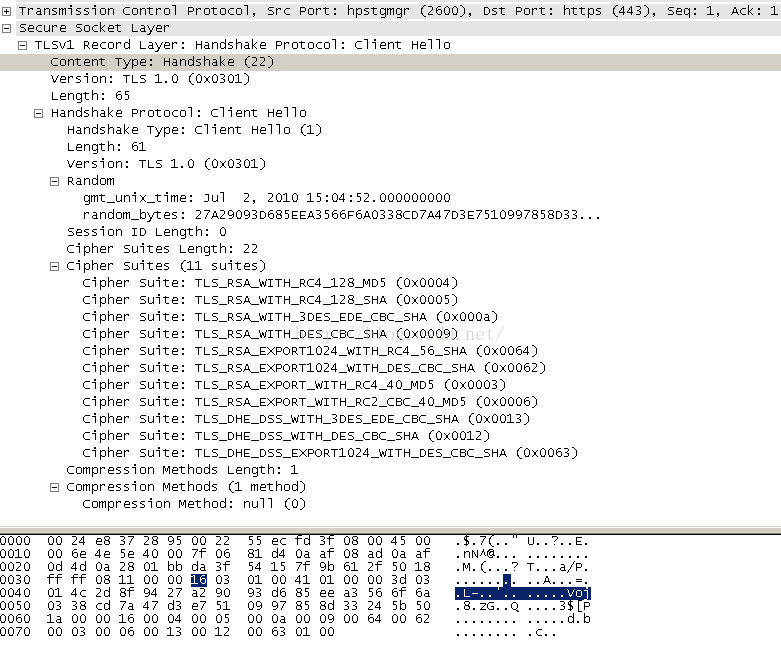
8.Server发送ChangeCipherSpec，指示Client从现在开始发送的消息都是加密过的

9.Server发送Finishd，包含了前面所有握手消息的hash，可以让client验证握手过程是否被第三方篡改，并且证明自己是Certificate密钥的拥有者，即证明自己的身份

三、抓包实际分析连接过程

下面从抓包数据来具体分析这一过程并说明各部分数据的作用以及如实现前面列出的握手的目标，当然了，最重要的还是说明为何这一过程是安全可靠的，第三方无法截获，篡改或者假冒。

**1.client发送ClientHello**



每一条消息都会包含有ContentType,Version,HandshakeType等信息。

ContentType指示SSL通信处于哪个阶段，是握手(Handshake)，开始加密传输(ChangeCipherSpec)还是正常通信(Application)等，见下表

|  |  |  |
| --- | --- | --- |
| **Hex** | **Dec** | **Type** |
| 0x14 | 20 | ChangeCipherSpec |
| 0x15 | 21 | Alert |
| 0x16 | 22 | Handshake |
| 0x17 | 23 | Application |

Version是TLS的版本，见下表

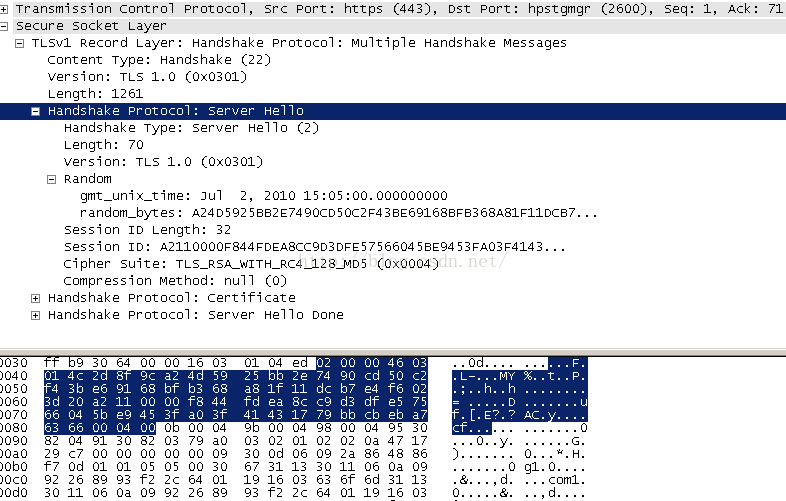
|  |  |  |
| --- | --- | --- |
| **Major Version** | **Minor Version** | **Version Type** |
| 3 | 0 | SSLv3 |
| 3 | 1 | TLS 1.0 |
| 3 | 2 | TLS 1.1 |
| 3 | 3 | TLS 1.2 |

Handshake Type是在handshanke阶段中的具体哪一步，见下表

|  |  |
| --- | --- |
| **Code** | **Description** |
| 0 | HelloRequest |
| 1 | ClientHello |
| 2 | ServerHello |
| 11 | Certificate |
| 12 | ServerKeyExchange |
| 13 | CertificateRequest |
| 14 | ServerHelloDone |
| 15 | CertificateVerify |
| 16 | ClientKeyExchange |
| 20 | Finished |

ClientHello附带的数据随机数据RN，会在生成session key时使用，Cipher suite列出了client支持的所有加密算法组合，可以看出每一组包含3种算法，一个是非对称算法，如RSA，一个是对称算法如DES，3DES，RC4等，一个是Hash算法，如MD5，SHA等，server会从这些算法组合中选取一组，作为本次SSL连接中使用。

**2.server回应ServerHello**



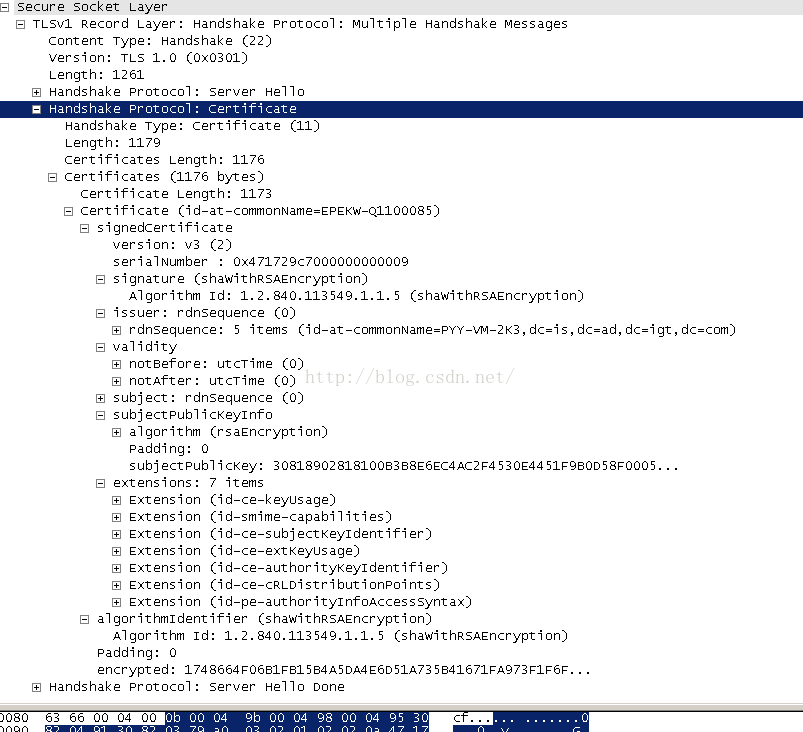
这里多了个session id,如果SSL连接断开，再次连接时，可以使用该属性重新建立连接，在双方都有缓存的情况下可以省略握手的步骤。

server端也会生成随机的RN，用于生成session key使用。

server会从client发送的Cipher suite列表中跳出一个，这里挑选的是RSA+RC4+MD5

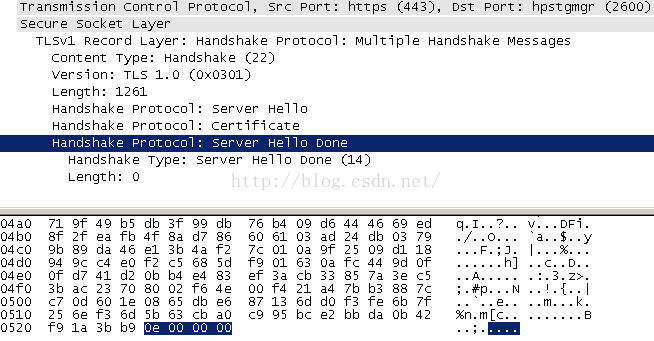
这次server共发送的3个handshake 消息:Serverhello，Certificate和ServerHelloDone，共用一个ContentType:Handshake

**3.server发送Certificate**

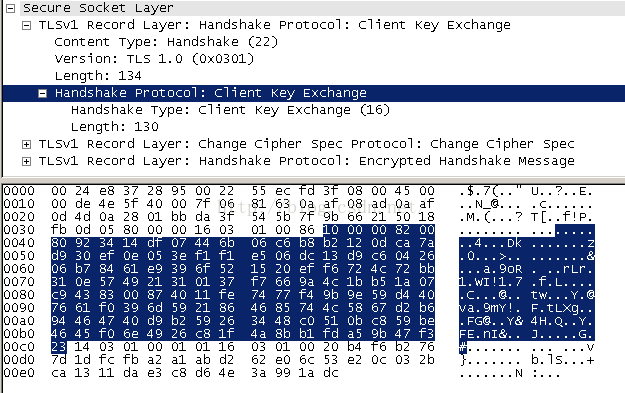


              server的证书信息，只包含public key，server将该public key对应的private key保存好，用于证明server是该证书的实际拥有者，那么如何验证呢？原理很简单：client随机生成一串数，用server这里的public key加密(显然是RSA算法)，发给server，server用private key解密后返回给client，client与原文比较，如果一致，则说明server拥有private key，也就说明与client通信的正是证书的拥有者，因为public key加密的数据，只有private key才能解密，目前的技术还没发破解。利用这个原理，也能实现session key的交换，加密前的那串随机数就可用作session key，因为除了client和server，没有第三方能获得该数据了。原理很简单，实际使用时会复杂很多，数据经过多次hash，伪随机等的运算，前面提到的client和server端得RN都会参与计算。

**4.Server发送ServerHelloDone**

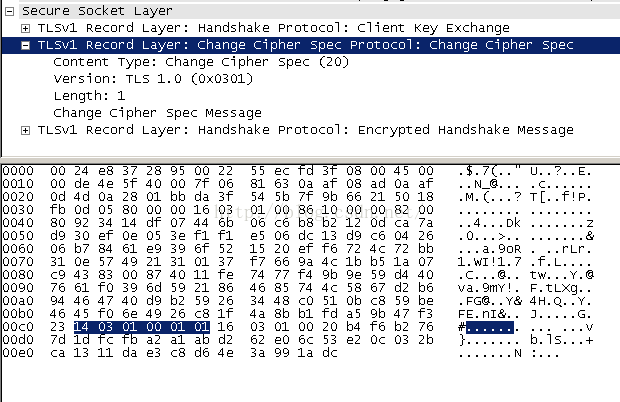


**5.Client发送ClientKeyExchange**



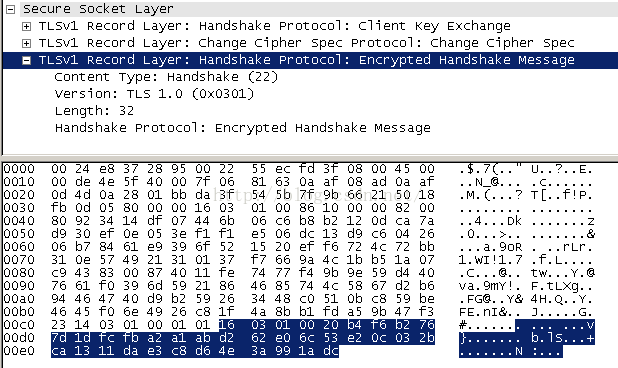
client拿到server的certificate后，就可以开始利用certificate里的public key进行session key的交换了。从图中可以看出，client发送的是130字节的字节流，显然是加过密的。client随机生成48字节的Pre-master secret，padding后用public key加密就得到这130字节的数据发送给server，server解密也能得到Pre-master secret。双方使用pre-master secret, "master secret"常量字节流，前期交换的server端RN和client的RN作为参数，使用一个伪随机函数PRF，其实就是hash之后再hash，最后得到48字节的master secret。master secret再与"key expansion"常量，双方RN经过伪随机函数运算得到key\_block，PRF伪随机函数可以可以仿佛循环输出数据，因此我们想得到多少字节都可以，就如Random伪随机函数，给它一个种子，后续用hash计算能得到无数个随机数，如果每次种子相同，得到的序列是一样的，但是这里的输入时48字节的master secret，2个28字节的RN和一个字符串常量，碰撞的可能性是很小的。得到key block后，算法，就从中取出session key，IV(对称算法中使用的初始化向量)等。client和server使用的session key是不一样的，但只要双方都知道对方使用的是什么就行了。这里会取出4个：client/server加密正文的key，client/server计算handshake数据hash的key。

**6.Client发送ChangeCipherSpec**



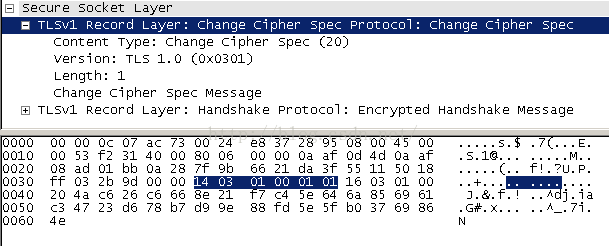
client指示Server从现在开始发送的消息都是加密过的。

**7.Client发送Finished**



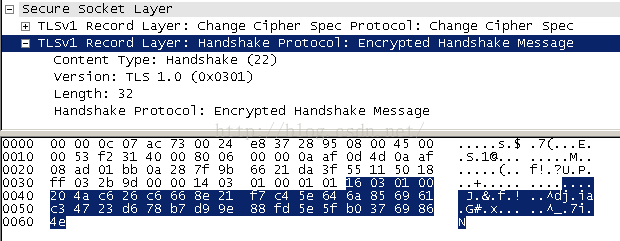
              client发送的加密数据，这个消息非常关键，一是能证明握手数据没有被篡改过，二也能证明自己确实是密钥的拥有者(这里是单边验证，只有server有certificate，server发送的Finished能证明自己含有private key，原理是一样的)。client将之前发送的所有握手消息存入handshake messages缓存，进行MD5和SHA-1两种hash运算，再与前面的master secret和一串常量"clientfinished"进行PRF伪随机运算得到12字节的verify data，还要经过改进的MD5计算得到加密信息。为什么能证明上述两点呢，前面说了只有密钥的拥有者才能解密得到pre-master key，master key，最后得到key block后，进行hash运算得到的结果才与发送方的一致。

**8.Server发送ChangeCipherSpec**



Server指示client从现在开始发送的消息都是加密过的。

**9.Server发送Finishd**



              与client发送Finished计算方法一致。server发送的Finished里包含hash给client，client会进行校验，如果通过，说明握手过程中的数据没有被第三方篡改过，也说明server是之前交换证书的拥有者。现在双方就可以开始后续通信，进入Application context了。