一、关于SSL、TLS与HTTPS的关系介绍

1. 什么是SSL、TLS

众所周知，真正的通信实际上是两台主机之间的进程在交换数据，而运输层作为整个网络最关键的从层次之一，扮演沿着向上层（应用层）提供通信服务的角色。想要剖析运输层的数据安全传输策略就一定无法绕开三个至关重要的协议，它们分别是HTTPS协议、SSL协议、TSL协议。SSL（Secure Sockets Layer）协议既安全套接字层协议，TLS（Transport Layer Security）协议即安全传输层协议。

2. 诞生背景

随着网上银行和电子商务的兴起，人们需要一种技术来保障Word Wide Web（WWW）通讯的安全，而当时现有的http协议一直是明文传输。在这一背景下Netscape（网景）公司于1990年开发出了SSL（Secure Sockets Layer）安全套接字层（此时的Netscape Navigator浏览器仍统治互联网浏览器市场）。该协议的主要任务是提供数据加密、身份验证和消息完整性验证服务。此时的SSL版本号为SSL 1.0,由于多方面原因第一个版本从未发布过。第二版既SSL 2.0则于1994年发布，但是由于Netscape的开发人员闭门造车基本没有与其他业内安全专家商讨，所以存在弱点，被认为是失败的协议，最终退出了历史舞台。时间来到了1995年我们的主角SSL 3.0终于发布，虽然与早先的版本协议名称相似，但是SSL 3.0被完全重新设计，该协议的主要任务不变且设计沿用至今。HTTP作为第一个使用SSL保证安全的应用层协议，HTTP over SSL(HTTPS)应运而生，后来HTTPS在RFC2818被标准化，这一设计也就沿用至今。但SSL的进化没有停止，现如今的HTTPS实际上是HTTP over SSL/TSL，TSL实际上可以算作SSL的升级版本但是由于IETF标准化被更名为TSL（Transport Layer Security，传输层安全）。最早的TLS 1.0版本与1999年发布且与SSL 3.0没有太大区别。TLS 1.1与2006年发布，时隔多年但也只是修复了一些关键的安全问题。TLS 1.2于2008年发布，该版本添加了对已验证加密的支持，并且使协议完全弹性化，该版本被沿用至今。最新的TLS 1.3版本针对安全强化及效率提升方面进行了大量修改，相继推出了20多个草案版本并且即将完成最终的标准化，将会得到广泛的支持。

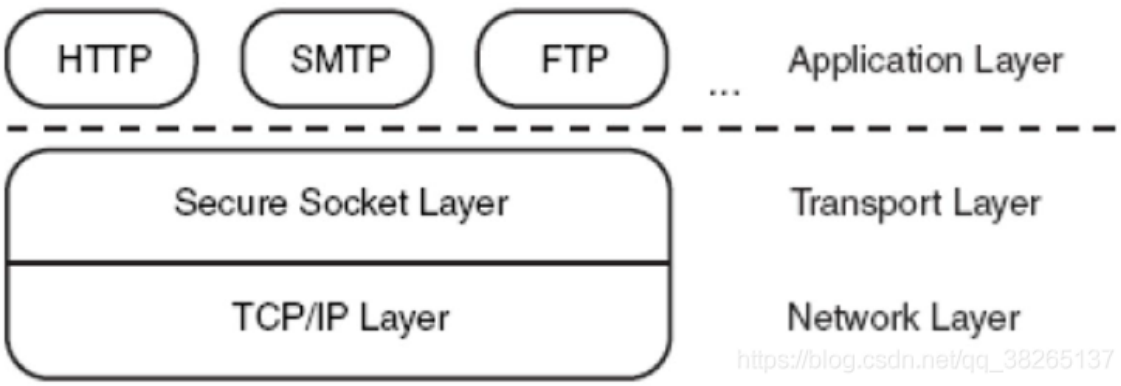
3. HTTPS、SSL与TLS的关系

SSL严格意义上将SSL是介于网络层协议和应用层协议之间的一种协议层。通过与HTTP协议搭配产生了HTTPS协议，您可以理解为HTTP+SSL=HTTPS（阅读下文您将理解为什么网络专家要这么设计）。现如今SSL经过三代更新，在SSLv3.0之后正式更名为TLS1.0。也可以理解为TLS1.0实际上是SSLv3.1。

二、SSL协议详解

SSL是一个不依赖于平台和运用程序的协议，位于TCP/IP协议与各种应用层协议之间，为数据通信提高安全支持。

SSL 和 TCP/IP 示意图



1. SSL加密知名协议

HTTP over SSL:

简写https，加密网页浏览是设计SSL的初衷，HTTP也是第一个使用SSL保障安全的应用层协议。

当Netscape在它的Navigator里面运用HTTP over SSL的时候，使用https://来标识HTTP over SSL，因此常见的https的全称就是HTTP over SSL。后来HTTPS在RFC2818被标准化。HTTPS工作在443端口，而HTTP默认工作在80端口。

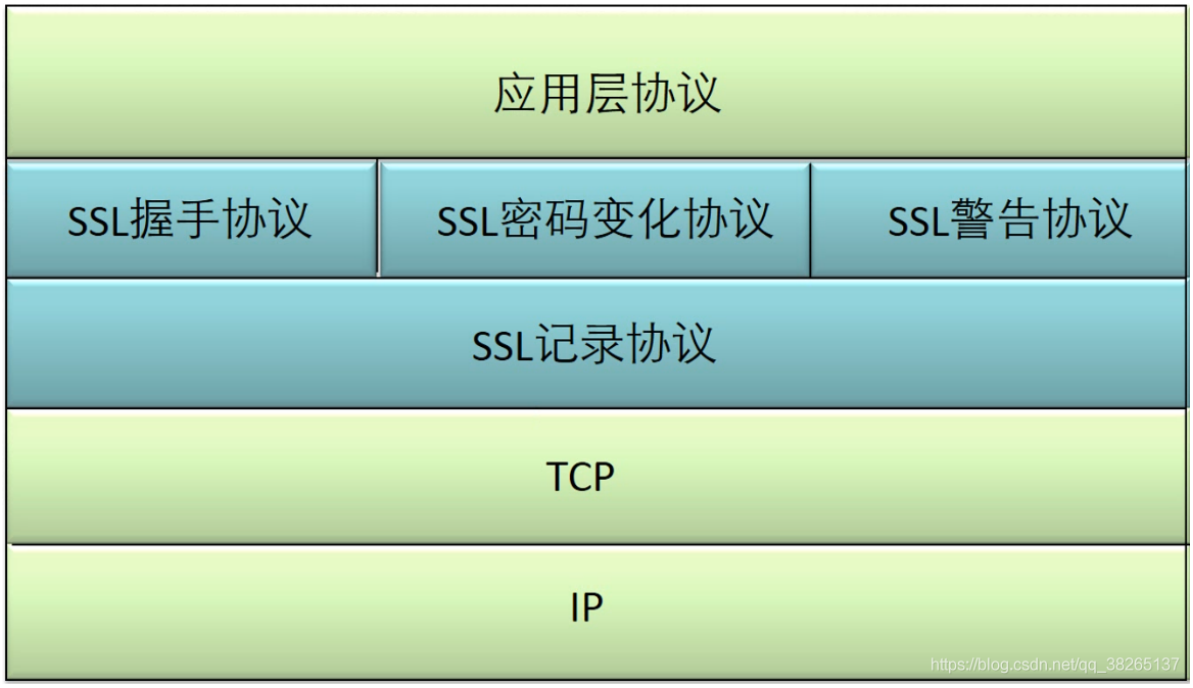
Email over SSL:

类似于HTTP over SSL，邮件协议例如：

- SMTP，POP3、IMAP也能支持SSL。
- SMTP over TLS的标准文档在RFC2487
- POP3和IMAP over TLS的标准化文档在RFC2595.

2. SSL原理详解

2.1 SSL协议结构:



SSL的体系结构中包含两个协议子层，其中底层是SSL记录协议层（SSL Record Protocol Layer）；高层是SSL握手协议层（SSL HandShake Protocol Layer）。

- SSL记录协议层的作用是为高层协议提供基本的安全服务。SSL纪录协议针对HTTP协议进行了特别的设计，使得超文本的传输协议HTTP能够在SSL运行。纪录封装各种高层协议，具体实施压缩解压缩、加密解密、计算和校验MAC等与安全有关的操作。
- SSL握手协议层包括SSL握手协议（SSL HandShake Protocol）、SSL密码参数修改协议（SSL Change Cipher Spec Protocol）和SSL告警协议（SSL Alert Protocol）。握手层的这些协议用于SSL管理信息的交换，允许应用协议传送数据之间相互验证，协商加密算法和生成密钥等。

SSL握手协议的作用是协调客户和服务器的状态，使双方能够达到状态的同步。

2.2 SSL建立阶段

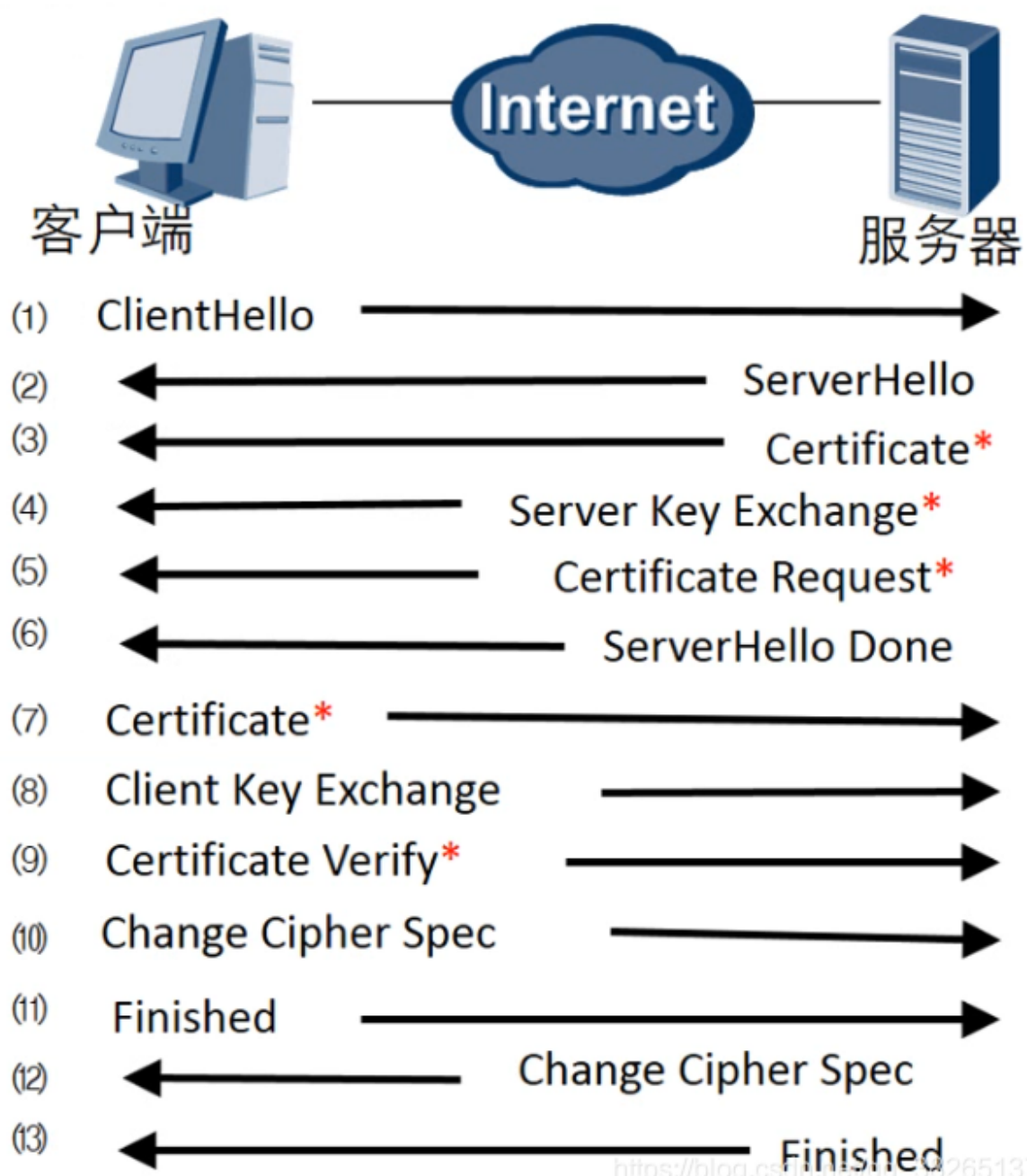
可以分为两个大阶段：

（1）SSL建立的第一阶段：Handshake phase（握手阶段）：

- 协商加密算法
- 认证服务器
- 建立用于加密和MAC（Message Authentication Code）用的密钥

（2）SSL建立第二阶段：Secure data transfer phase（安全的数据传输阶段）：

- 在已经建立的SSL连接里安全的传输数据。

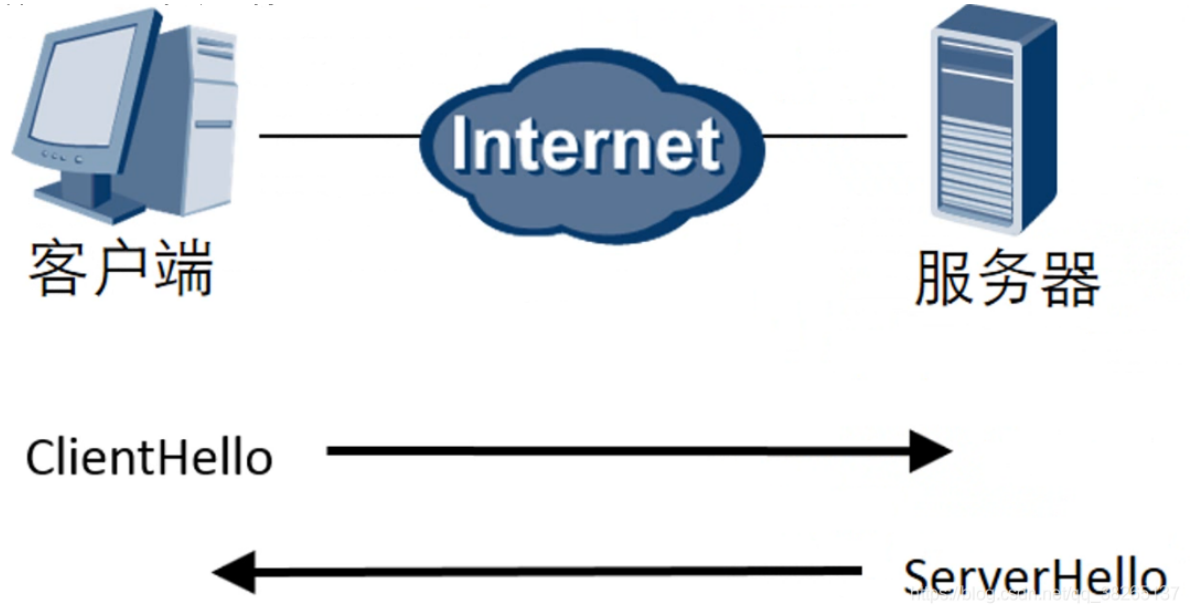


图：SSL建立总过程

SSL的建立过程总共有13个包，第一次建立至少需要9个包。

2.2.1 SSL建立第一阶段：

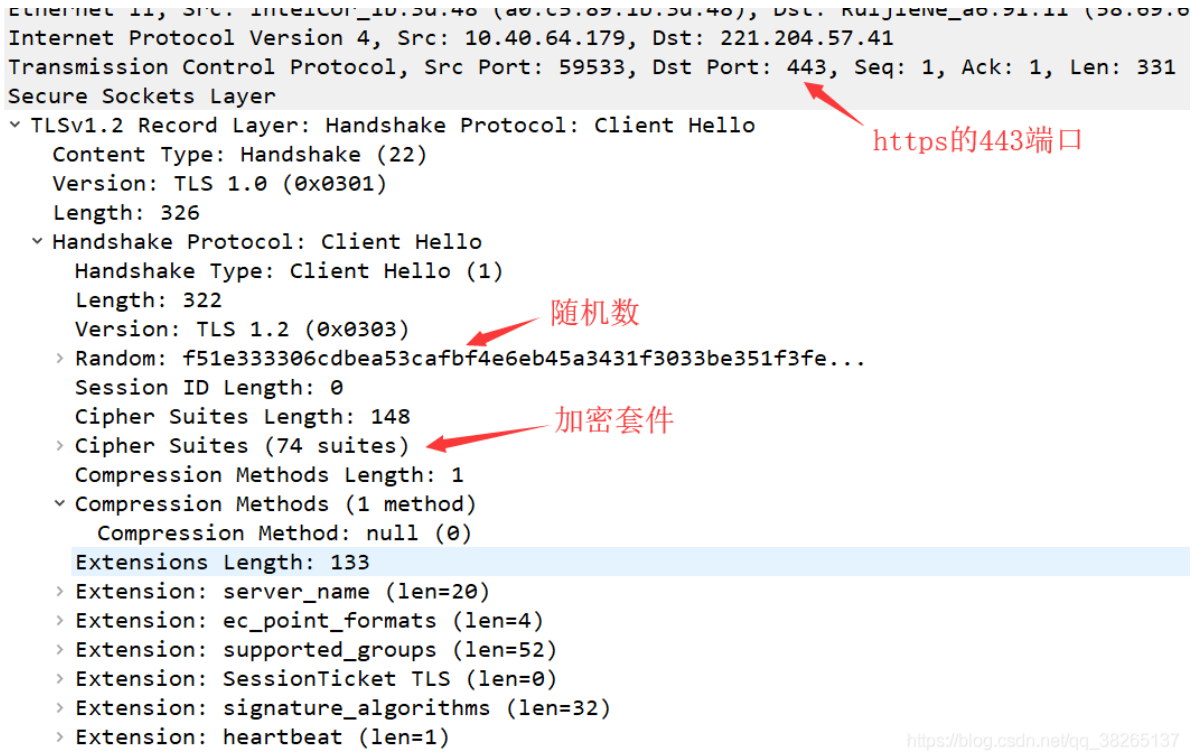
客户端首先发送ClientHello消息到服务端，服务端收到ClientHello消息后，再发送ServerHello消息回应客户端。



图：SSL建立第一阶段报文交换示意图

ClientHello

握手第一步是客户端向服务端发送 Client Hello 消息，这个消息里包含了一个客户端生成的随机数 **Random1**、客户端支持的加密套件（Support Ciphers）和 SSL Version 等信息。



图：ClinetHello报文抓包示例

ClientHello中涉及到的消息具体如下：

- 客户端版本

  按优先级列出客户端支持的协议版本，首选客户端希望支持的最新协议版本。

- 客户端随机数Random

- 会话ID（Session id）

如果客户端第一次连接到服务器，那么这个字段就会保持为空。上图中该字段为空，说明这是第一次连接到服务器。

如果该字段不为空，说明以前是与服务器有连接的，在此期间，服务器将使用Session ID映射对称密钥，并将Session ID存储在客户　　端浏览器中，为映射设置一个时间限。如果浏览器将来连接到同一台服务器（在时间到期之前），它将发送Session ID，服务器将对　映射的Session ID进行验证，并使用以前用过的对称密钥来恢复Session，这种情况下不需要完全握手。也叫作SSL会话恢复。后面会有介绍。

- 加密套件：

客户端会给服务器发送自己已经知道的密码套件列表，这是由客户按优先级排列的，但完全由服务器来决定发送与否。TLS中使用的　　密码套件有一种标准格式。上面的报文中，客户端发送了74套加密套件。服务端会从中选出一种来作为双方共同的加密套件。

- 压缩方法：

为了减少带宽，可以进行压缩。但从成功攻击TLS的事例中来看，其中使用压缩时的攻击可以捕获到用HTTP头发送的参数，这个攻击　可以劫持Cookie，这个漏洞我们称为CRIME。从TLS 1.3开始，协议就禁用了TLS压缩。

- 扩展包：

其他参数（如服务器名称，填充，支持的签名算法等）可以作为扩展名使用。

这些是客户端问候的一部分，如果已收到客户端问候，接下来就是服务器的确认，服务器将发送服务器问候。

ServerHello

收到**客户端问候**之后服务器必须发送**服务器问候**信息，服务器会检查指定诸如TLS版本和算法等客户端问候的条件，如果服务器接受并支持所有条件，它将发送其证书以及其他详细信息，否则，服务器将发送握手失败消息。

如果接受，第二步是服务端向客户端发送 Server Hello 消息，这个消息会从 Client Hello 传过来的 Support Ciphers 里确定一份加密套件，这个套件决定了后续加密和生成摘要时具体使用哪些算法，另外还会生成一份**随机数 Random2**。注意，至此客户端和服务端都拥有了**两个随机数（Random1+Random2）**，这两个随机数会在后续生成对称秘钥时用到。

图：ServerHello报文抓包

ServerHello中涉及到的具体参数：

- 服务器版本Version：

  服务器会选择客户端支持的最新版本。

- 服务器随机数Random：

  服务器和客户端都会生成32字节的随机数。用来创建加密密钥。

- 加密套件：

  服务器会从客户端发送的加密套件列表中选出一个加密套件。

- 会话ID（Session ID）：

服务器将约定的Session参数存储在TLS缓存中，并生成与其对应的Session id。它与Server Hello一起发送到客户端。客户端可以写　入约定的参数到此Session id，并给定到期时间。客户端将在Client Hello中包含此id。如果客户端在此到期时间之前再次连接到服务　器，则服务器可以检查与Session id对应的缓存参数，并重用它们而无需完全握手。这非常有用，因为服务器和客户端都可以节省大量的计算成本。

在涉及亚马逊和谷歌等流量巨大的应用程序时，这种方法存在缺点。每天都有数百万人连接到服务器，服务器必须使用Session密钥　保留所有Session参数的TLS缓存。这是一个巨大的开销。

为了解决这个问题，在扩展包里加入了Session Tickets, 在这里，客户端可以在client hello中指定它是否支持Session Ticket。然后，　服务器将创建一个新的会话票证(Session Ticket)，并使用只有服务器知道的经过私钥加密的Session参数。它将存储在客户端上，因　此所有Session数据仅存储在客户端计算机上，但Ticket仍然是安全的，因为该密钥只有服务器知道。

此数据可以作为名为Session Ticket的扩展包含在Client Hello中。

- 压缩算法：

  如果支持，服务器将同意客户端的首选压缩方法。

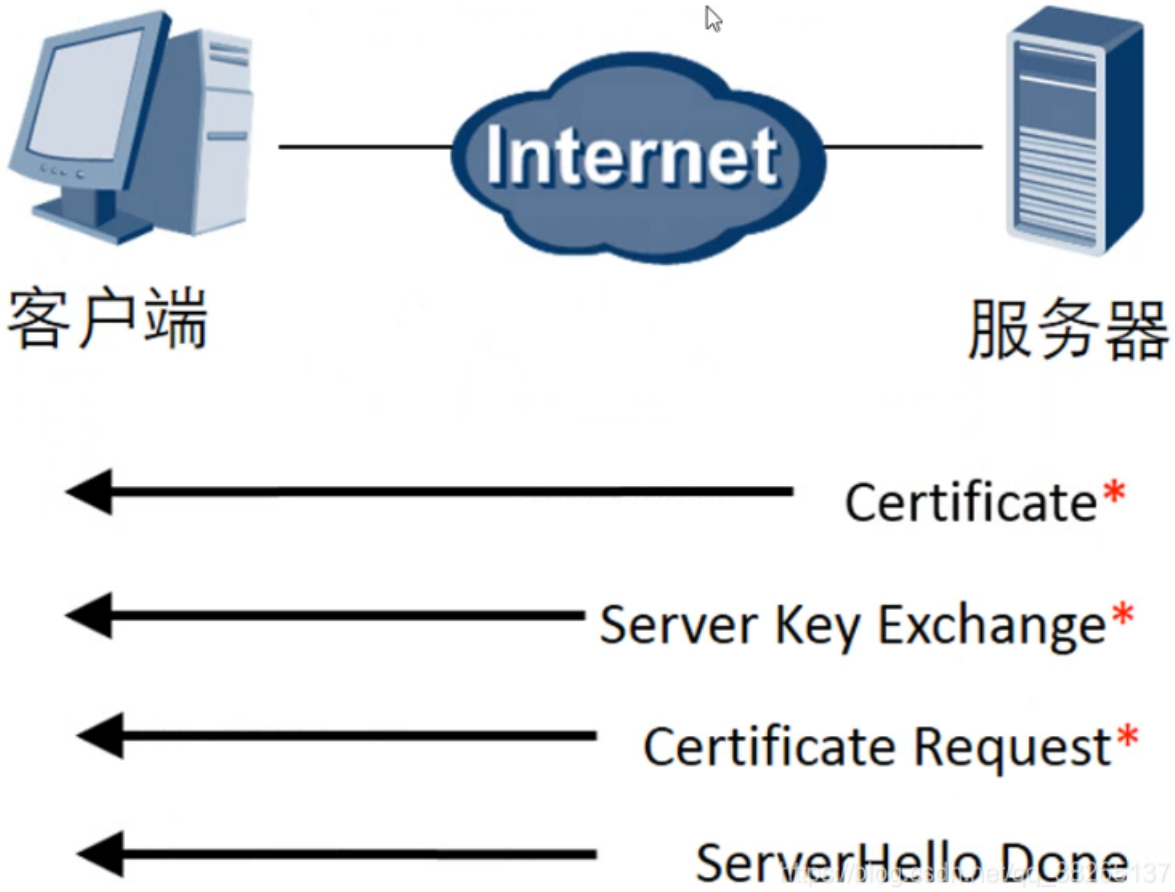
- 扩展包

这个阶段之后，**客户端服务端**知道了下列内容：

1. SSL版本
2. 密钥交换、信息验证和加密算法
3. 压缩方法
4. 有关密钥生成的两个随机数。

2.2.2 SSL建立第二阶段：

服务器向客户端发送消息。



图：SSL建立第二阶段报文交换示意图

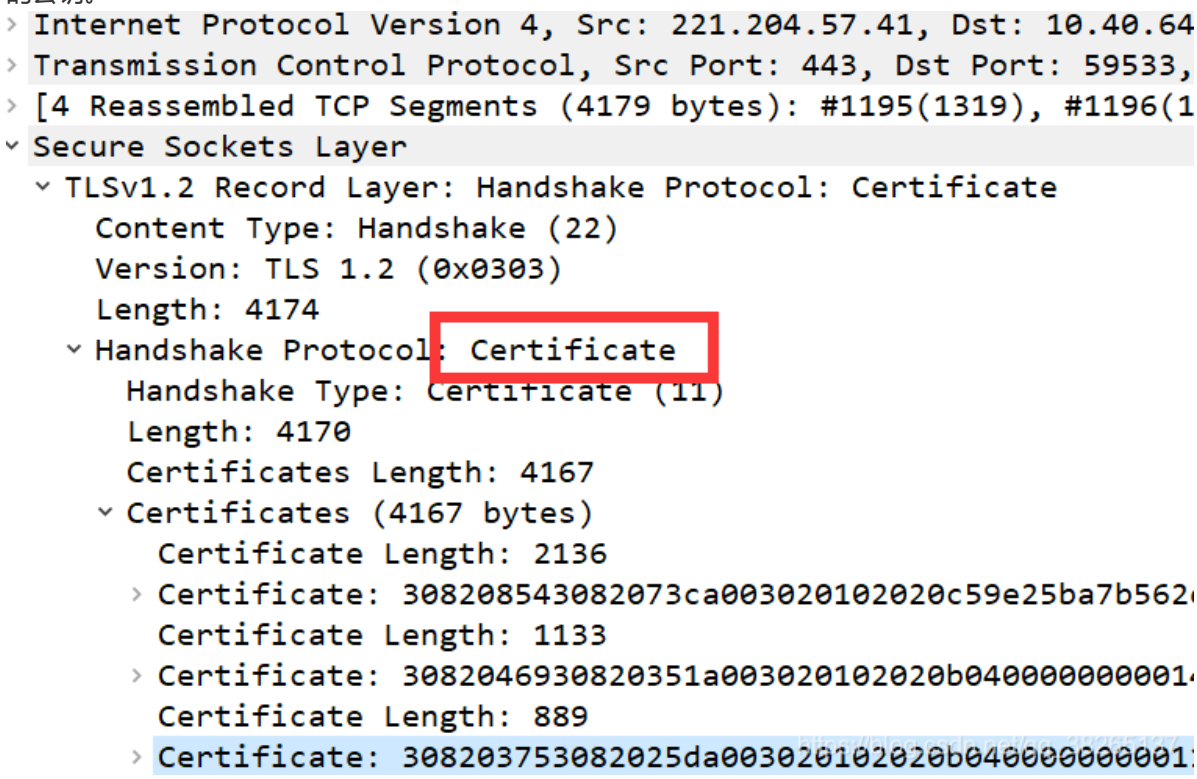
服务器启动SSL握手第2阶段，也是本阶段所有消息的唯一发送方，客户机是所有消息的唯一接收方。该阶段分为4步：

1. 证书：服务器将数字证书和到根CA整个链发给客户端，使客户端能用服务器证书中的服务器公钥认证服务器。
2. 服务器密钥交换（可选）：这里视密钥交换算法而定
3. 证书请求：服务端可能会要求客户自身进行验证。
4. 服务器握手完成：第二阶段的结束，第三阶段开始的信号

Certificate消息（可选）—第一次建立必须要有证书

一般情况下，除了会话恢复时不需要发送该消息，在SSL握手的全流程中，都需要包含该消息。消息包含一个X.509证书，证书中包含公钥，发给客户端用来验证签名或在密钥交换的时候给消息加密。

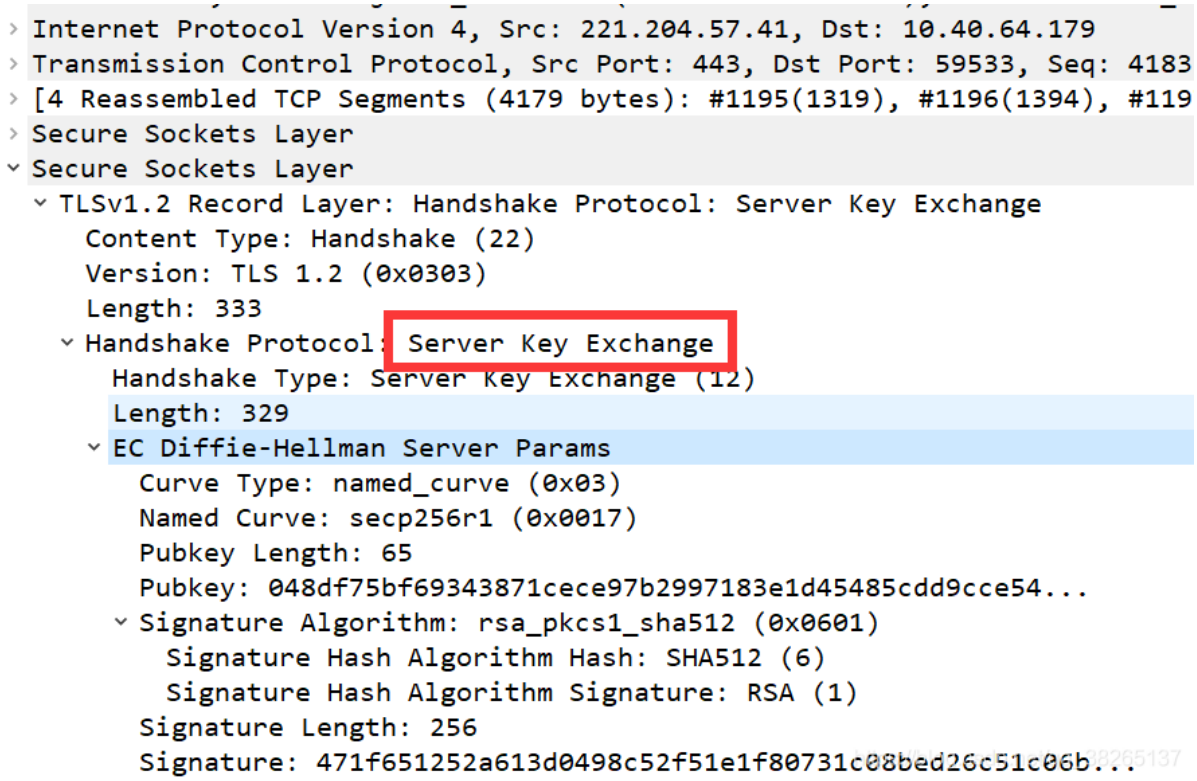
这一步是服务端将自己的证书下发给客户端，让客户端验证自己的身份，客户端验证通过后取出证书中的公钥。



图：服务器给客户端发送的证书报文

Server Key Exchange（可选）

根据之前在ClientHello消息中包含的CipherSuite信息，决定了密钥交换方式（例如RSA或者DH），因此在Server Key Exchange消息中便会包含完成密钥交换所需的一系列参数。



图：Server Key Exchange报文

因为这里是DH算法，所以需要发送服务器使用的DH参数。RSA算法不需要这一步。

在Diffie-Hellman中，客户端无法自行计算预主密钥；双方都有助于计算它，因此客户端需要从服务器获取Diffie-Hellman公钥。

由上图可知，此时密钥交换也由签名保护。

Certificate Request（可选）------可以是单向的身份认证，也可以双向认证

这一步是可选的，如果在对安全性要求高的常见可能用到。服务器用来验证客户端。服务器端发出Certificate Request消息，要求客户端发他自己的证书过来进行验证。该消息中包含服务器端支持的证书类型（RSA、DSA、ECDSA等）和服务器端所信任的所有证书发行机构的CA列表，客户端会用这些信息来筛选证书。

Server Hello Done

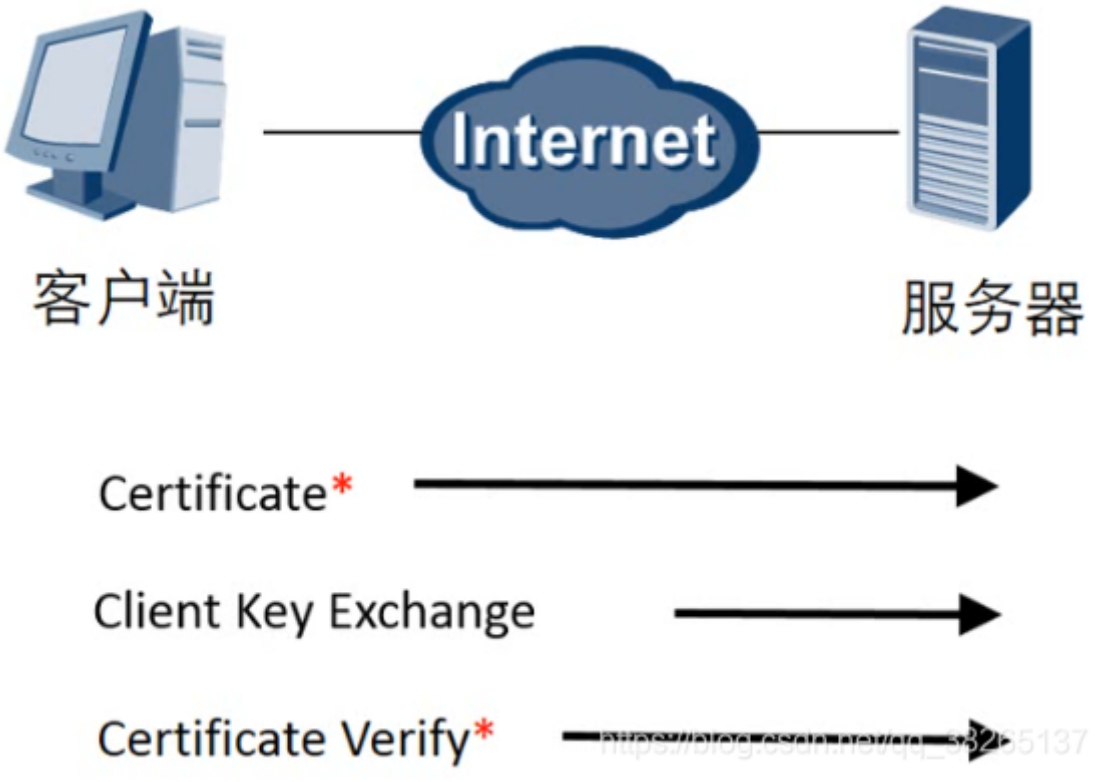
该消息表示服务器已经将所有信息发送完毕，接下来等待客户端的消息。

```
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 4
    Handshake Protocol: Server Hello Done
      Handshake Type: Server Hello Done (14)
      Length: 0
```


图 Server Hello Done报文

2.2.3 SSL建立第三阶段：

客户端收到服务器发送的一系列消息并解析后，将本端相应的消息发送给服务器。



图：SSL建立第三阶段报文交换示意图

客户机启动SSL握手第3阶段，它是本阶段所有消息的唯一发送方，服务器是所有消息的唯一接收方。该阶段分为3步：

1.证书（可选）：为了对服务器证明自身，客户要发送一个证书信息，这是可选的，在IIS中可以配置强制客户端证书认证。

2.客户机密钥交换（Pre-master-secret）：这里客户端将预备主密钥发送给服务端，注意这里会使用服务端的公钥进行加密。

3.证书验证（可选），对预备秘密和随机数进行签名，证明拥有（a）证书的公钥。

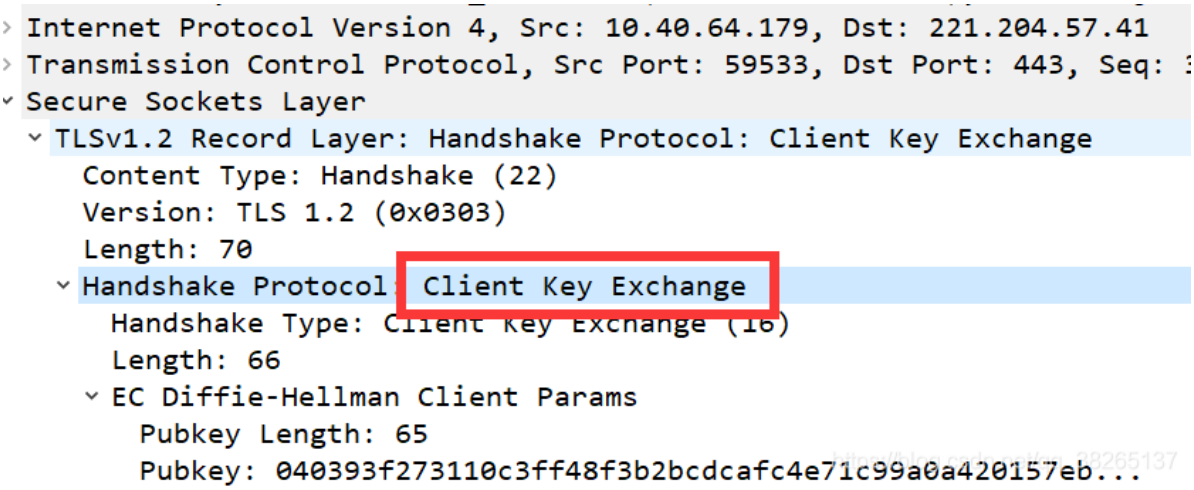
Certificate（可选）

如果在第二阶段服务器端要求发送客户端证书，客户端便会在该阶段将自己的证书发送过去。服务器端在之前发送的Certificate Request消息中包含了服务器端所支持的证书类型和CA列表，因此客户端会在自己的证书中选择满足这两个条件的第一个证书发送过去。若客户端没有证书，则发送一个no_certificate警告。

Client Key exchange

根据之前从服务器端收到的随机数，按照不同的密钥交换算法，算出一个pre-master，发送给服务器，服务器端收到pre-master算出main master。而客户端当然也能自己通过pre-master算出main master。如此以来**双方就都算出并拥有了对称密钥。**

如果是RSA算法，会生成一个48字节的随机数，然后用server的公钥加密后再放入报文中。如果是DH算法，这次发送的就是客户端的DH参数，之后服务器和客户端根据DH算法，各自计算出相同的pre-master secret.



图：Clinet Key exchange报文

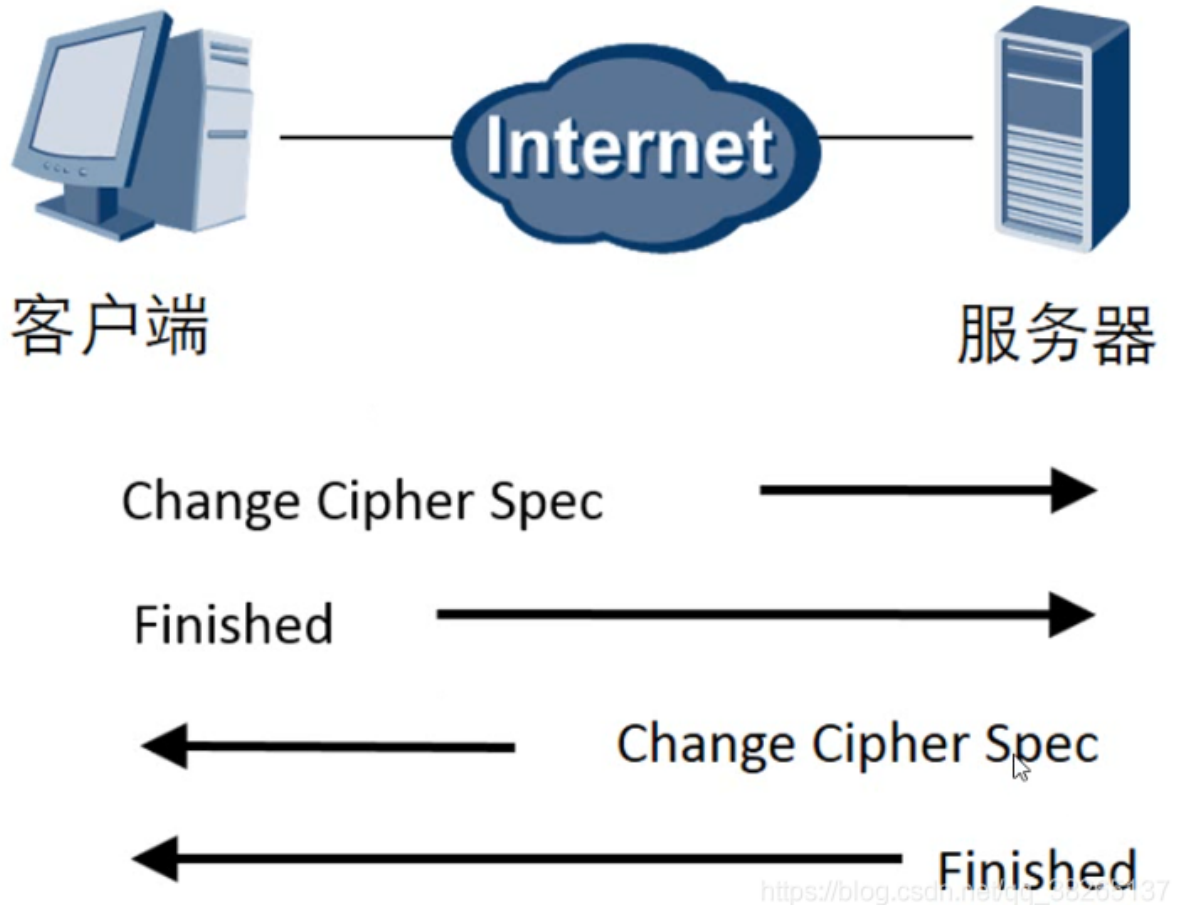
本消息在给服务器发送的过程中，使用了服务器的公钥加密。服务器用自己的私钥解密后才能得到pre-master key.（向服务器证明自己的确持有客户端证书私钥。）

Certificate verify（可选）

只有在客户端发送了自己证书到服务器端，这个消息才需要发送。其中包含一个签名，对从第一条消息以来的所有握手消息的HMAC值（用master_secret）进行签名。

2.2.4 SSL建立第四阶段：

完成握手协议，建立SSL连接。



客户机启动SSL握手第4阶段，使服务器结束。该阶段分为4步，前2个消息来自客户机，后2个消息来自服务器。

建立起一个安全的连接，客户端发送一个Change Cipher Spec消息，并且把协商得到的CipherSuite拷贝到当前连接的状态之中。然后，客户端用新的算法、密钥参数发送一个Finished消息，这条消息可以检查密钥交换和认证过程是否已经成功。其中包括一个校验值，对客户端整个握手过程的消息进行校验。服务器同样发送Change Cipher Spec消息和Finished消息。握手过程完成，客户端和服务器可以交换应用层数据进行通信。

ChangeCipherSpec：

> 编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送（ChangeCipherSpec是一个独立的协议，体现在数据包中就是一个字节的数据，用于告知服务端，客户端已经切换到之前协商好的加密套件（Cipher Suite）的状态，准备使用之前协商好的加密套件加密数据并传输了）。

```
› Internet Protocol Version 4, Src: 10.40.64.179, Dst: 221.204.57.41
› Transmission Control Protocol, Src Port: 59533, Dst Port: 443, Seq: 33
⌄ Secure Sockets Layer
  › TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
  ⌄ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spe
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
```


图：Cipher Spec Message报文

Clinet Finished:

客户端握手结束通知, 表示客户端的握手阶段已经结束。这一项同时也是前面发送的**所有内容的hash值**，用来供服务器校验（使用HMAC算法计算收到和发送的所有握手消息的摘要，然后通过RFC5246中定义的一个伪函数PRF计算出结果，加密后发送。此数据是为了在正式传输应用数据之前对刚刚握手建立起来的加解密通道进行验证。）

Server Finished:

服务端握手结束通知。

1. 使用私钥解密加密的Pre-master数据，基于之前(Client Hello 和 Server Hello)交换的两个明文随机数 Random1 和 Random2，计算得到协商密钥:enc_key=Fuc(Random1, Random2, Pre-Master);
2. 计算之前所有接收信息的 hash 值，然后解密客户端发送的 encrypted_handshake_message，验证数据和密钥正确性;
   发送一个 ChangeCipherSpec（告知客户端已经切换到协商过的加密套件状态，准备使用加密套件和 enc_key加密数据了）
3. 服务端也会使用 enc_key 加密一段 Finish 消息发送给客户端，以验证之前通过握手建立起来的加解密通道是否成功。

> 根据之前的握手信息，如果客户端和服务端都能对`Finish`信息进行正常加解密且消息正确的被验证，则说明握手通道已经建立成功，接下来，双方可以使用上面产生的`Session Secret`对数据进行加密传输了。

2.3 消息验证代码（HMAC）和TLS数据完整性

当服务器或客户端使用主密钥加密数据时，它还会计算明文数据的校验和（哈希值），这个校验和称为消息验证代码（MAC）。然后在发送之前将MAC包含在加密数据中。密钥还被用于从数据中生成MAC，以确保传输过程中攻击者无法从数据中生成相同的MAC，故而MAC被称为HMAC（哈希消息认证码）。另一方面，在接收到消息时，解密器将MAC与明文分开，然后用它的密钥计算明文的校验和，并将其与接收到的MAC进行比较，如果匹配，那我们就可以得出结论：数据在传输过程中没有被篡改。

2.4 几个重要的secret key

2.4.1 PreMaster secret

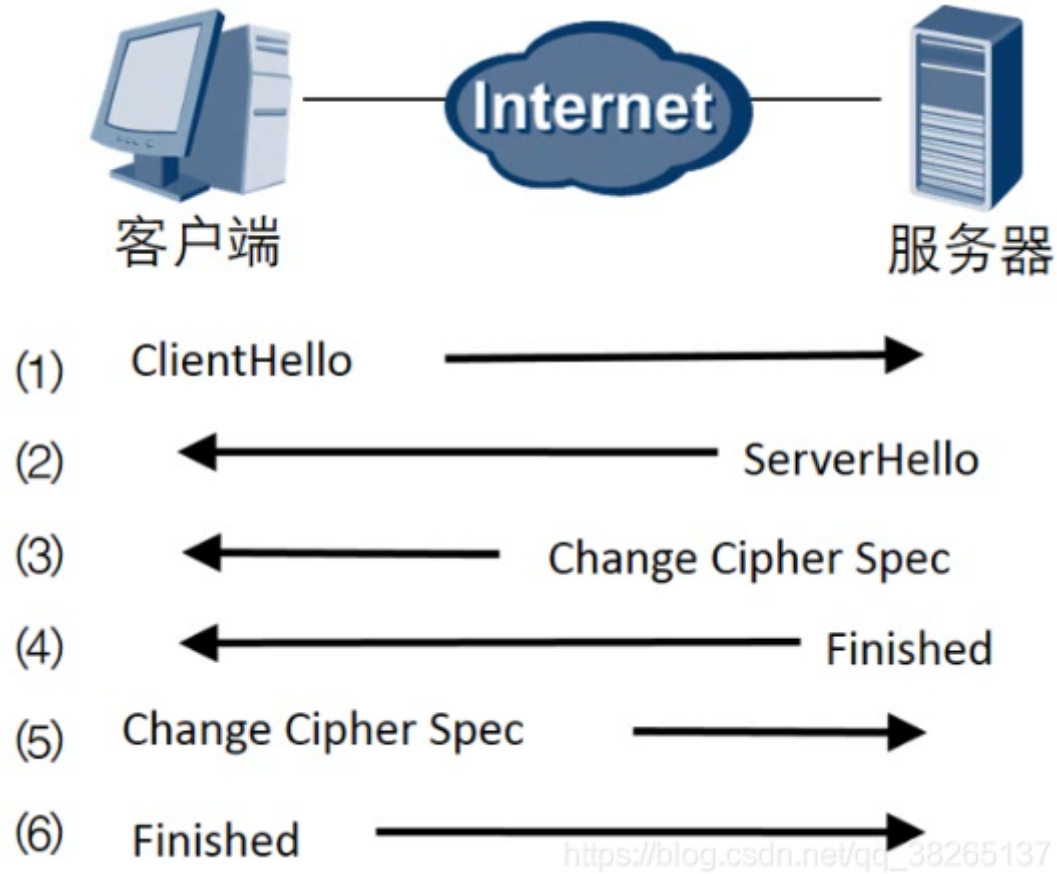
PreMaster Secret是在客户端使用RSA或者Diffie-Hellman等加密算法生成的。它将用来跟服务端和客户端在Hello阶段产生的随机数结合在一起生成 Master Secret。PreMaster secret前两个字节是TLS的版本号，这是一个比较重要的用来核对握手数据的版本号。服务端需要对密文中解密出来对的PreMaster版本号跟之前Client Hello阶段的版本号进行对比，如果版本号变低，则说明被串改，则立即停止发送任何消息。

2.4.2 Master secret

由于最后通过交换，客户端和服务端都会有Pre-master和随机数，这个随机数将作为后面产生Master secret的种子，结合PreMaster secret，客户端和服务端将计算出同样的Master secret。

2.5 SSL会话恢复

会话恢复是指只要客户端和服务器已经通信过一次，它们就可以通过会话恢复的方式来跳过整个握手阶段二、阶段三直接进行数据传输。



图：SSL会话恢复过程

SSL采用会话恢复的方式来减少SSL握手过程中造成的巨大开销。

为了加快建立握手的速度，减少协议带来的性能降低和资源消耗(具体分析在后文)，TLS 协议有**两类会话缓存机制**：

```
1.会话标识 session ID: 由服务器端支持，协议中的标准字段，因此基本所有服务器都支持，服务器端保存会话ID以及协商的通信信息，Nginx 中1M 内存约可以保存4000个 session ID 机器相关信息，占用服务器资源较多;

2.会话记录 session ticket :t需要服务器和客户端都支持，属于一个扩展字段，支持范围约60%(无可靠统计与来源)，将协商的通信信息加密之后发送给客户端保存，密钥只有服务器知道，占用服务器资源很少。

二者对比，主要是保存协商信息的位置与方式不同，类似与 http 中的 session 与 cookie。二者都存在的情况下，(nginx 实现)优先使用 session_ticket。
```

2.5.1 会话恢复具体过程（Session ID机制）：

1. 如果客户端和服务器之间曾经建立了连接，服务器会在握手成功后返回 session ID，并保存对应的通信参数在服务器中;
   如果客户端再次需要和该服务器建立连接，则在 client_hello 中 session ID 中携带记录的信息，发送给服务器;
2. 服务器根据收到的 session ID 检索缓存记录，如果没有检索到，说明缓存过期，则按照正常的握手过程进行;
3. 如果检索到对应的缓存记录，则返回 change_cipher_spec 与 encrypted_handshake_message 信息，两个信息作用类似，encrypted_handshake_message 包含当前的通信参数与 master_secret 的hash 值;

4. 如果客户端能够验证通过服务器加密数据，则客户端同样发送 change_cipher_spec 与 encrypted_handshake_message 信息;
   服务器验证数据通过，则握手建立成功，开始进行正常的加密数据通信。

2.5.2 会话恢复具体过程（ session ticket）：

1. 如果客户端和服务器之间曾经建立了连接，服务器会在 new_session_ticket 数据中携带加密的 session_ticket 信息，客户端保存;
   如果客户端再次需要和该服务器建立连接，则在 client_hello 中扩展字段 session_ticket 中携带加密信息，一起发送给服务器;
2. 服务器解密 sesssion_ticket 数据，如果能够解密失败，则按照正常的握手过程进行;
3. 如果解密成功，则返回 change_cipher_spec 与 encrypted_handshake_message 信息，两个信息作用与 session ID 中类似;
4. 如果客户端能够验证通过服务器加密数据，则客户端同样发送 change_cipher_spec与 encrypted_handshake_message 信息;
   服务器验证数据通过，则握手建立成功，开始进行正常的加密数据通信。