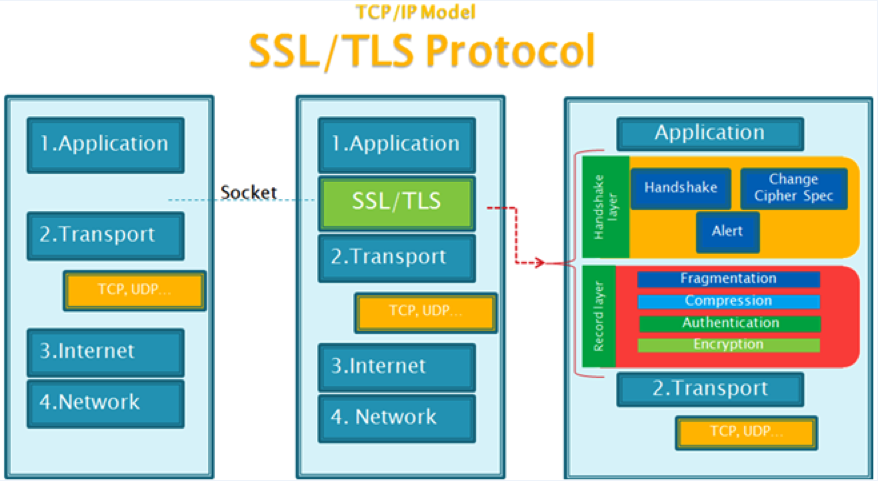
# SSL 原理说明

SSL/TLS是一个介于应用层（HTTP 协议）与传输层（TCP 协议）之间的一个可选协议，其协议架构可参照下图：  


当HTTP通信不使用 SSL/TLS 时，所有信息均以明文形式传播，会有以下风险：

* 窃听风险（eavesdropping）：第三方可以获得通信内容
* 篡改风险（tampering）：第三方可以修改通信内容
* 冒充风险（pretending）：第三方可以冒充他人身份参与通信

为了解决这些通信风险，SSL/TLS 协议应运而生。协议设计的目标为：

* 所有信息都是加密传播，第三方无法窃听。
* 具有校验机制，一旦被篡改，通信双方可以立即发现。
* 配备身份证书，防止身份被冒充。

目前，主流浏览器都已经支持了 SSL/TLS 的支持。

**1. SSL/TLS 协议基本运行过程**

SSL/TLS 协议的基本思路是采用公钥加密的方法。即客户端先向服务器端索要公钥，然后使用公钥加密信息并发送至服务器端；服务器收到密文后，用自己的私钥解密。

这里需要解决两个问题：

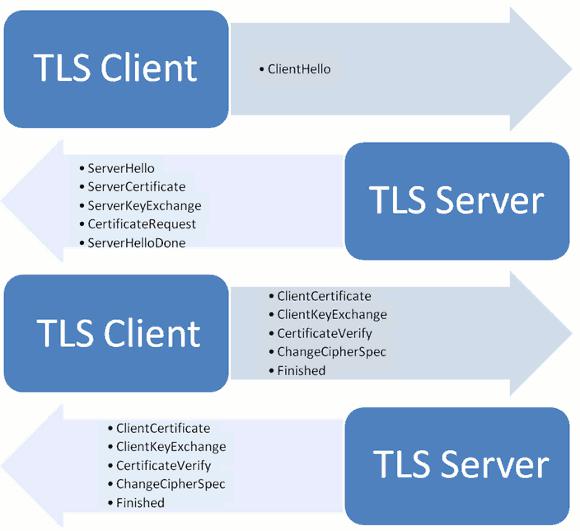
* 如何保证公钥不被篡改？  
  解决方法：将公钥放在数字证书中。只要证书是可信的，公钥就是可信的。
* 公钥加密计算量太大，如何减少耗用的时间？  
  解决方法：每一次对话（session），客户端和服务器端都生成一个"对话密钥"（session key），用它来加密信息。由于"对话密钥"是对称加密，所以运算速度非常快，而服务器公钥只用于加密"对话密钥"本身，这样就减少了加密运算的消耗时间。

SSL/TLS协议的基本过程是这样的：

* 客户端向服务器端索要并验证公钥
* 双方协商生成"对话密钥"
* 双方采用"对话密钥"进行加密通信。

上面过程的前两步，又称为"握手阶段"（handshake）。

**2. 握手阶段的详细过程**

  
"握手阶段"涉及四次通信且所有通信都是明文传输的。

**2.1. 客户端发出请求（ClientHello）**

客户端（通常是浏览器）先向服务器发出加密通信的请求，通常被称为ClientHello请求。

在这一步，客户端主要向服务器提供以下信息。

* 支持的协议版本，比如TLS 1.0
* 一个客户端生成的随机数，稍后用于生成"对话密钥"
* 支持的加密方法，比如RSA公钥加密
* 支持的压缩方法

这里需要注意的是，客户端发送的信息之中不包括服务器的域名。也就是说，理论上服务器只能包含一个网站，否则会分不清应该向客户端提供哪一个网站的数字证书。这就是为什么通常一台服务器只能有一张数字证书的。

对于虚拟主机的用户来说，这当然很不方便。2006年，TLS协议加入了一个 [Server Name Indication 扩展](http://tools.ietf.org/html/rfc4366)，允许客户端向服务器提供它所请求的域名。

**2.2. 服务器回应（SeverHello）**

服务器收到客户端请求后，向客户端发出回应，通常被称为SeverHello。服务器的回应包含以下内容：

* 确认使用的加密通信协议版本，比如TLS 1.0版本。如果浏览器与服务器支持的版本不一致，服务器将关闭加密通信
* 一个服务器生成的随机数，稍后用于生成"对话密钥"
* 确认使用的加密方法，比如RSA公钥加密
* 服务器证书

除了以上信息，若服务器需要确认客户端的身份，则会再包含一项请求，要求客户端提供"客户端证书"。比如，金融机构往往只允许认证客户连入自己的网络，就会向正式客户提供USB密钥，里面就包含了一张客户端证书。

**2.3. 客户端回应**

客户端收到服务器回应以后会首先验证服务器证书。如果证书不是由可信机构颁布、证书中的域名与实际域名不一致或者证书已经过期，就会向访问者显示一个警告，由其选择是否还要继续通信。  
如果证书没有问题，客户端就会从证书中取出服务器的公钥并向服务器发送以下信息：

* 一个随机数。该随机数用服务器公钥加密，防止被窃听。
* 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。
* 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。

上面第一项的随机数，是整个握手阶段出现的第三个随机数，又称"pre-master key"。有了它以后，客户端和服务器就同时有了三个随机数，接着双方就用事先商定的加密方法，各自生成本次会话所用的同一把"会话密钥"。

对于RSA密钥交换算法来说，pre-master-key本身就是一个随机数，再加上hello消息中的随机，三个随机数通过一个密钥导出器最终导出一个对称密钥。

pre-master的存在原因是SSL协议不信任每个主机都能产生完全随机的随机数。客户端和服务器加上pre-master三个随机数一同生成的密钥不容易被猜出了。因为一个伪随机可能完全不随机，可是三个伪随机则十分接近随机。

此外，如果前一步，服务器要求客户端证书，客户端会在这一步发送证书及相关信息。

**2.4. 服务器的最后回应**

服务器收到客户端的第三个随机数pre-master-key之后，计算生成本次会话所用的"会话密钥"。然后，向客户端最后发送如下信息：

* 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。
* 服务器握手结束通知，表示服务器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，用来供客户端校验。

至此，整个握手阶段全部结束。接下来，客户端与服务器进入加密通信，完全是使用HTTP协议，只不过用"会话密钥"加密内容。

使用一个简单的例子来说明上面的内容：假设A与B通信，A是SSL客户端，B是SSL服务器端，加密后的消息放在方括号[]里，以突出明文消息的区别。双方处理动作的说明用圆括号（）括起。

* A：

“我想和你安全的通话，我这里的对称加密算法有DES,RC5；密钥交换算法有RSA和DH；摘要算法有MD5和SHA。”

* B：

“我们用DES－RSA－SHA这对组合好了。这是我的证书，里面有我的名字和公钥，你拿去验证一下我的身份。”

把证书发给A。

“目前没有别的可说的了。”

* A：

查看证书上B的名字是否无误，并通过手头早已有的CA的证书验证了B的证书的真实性，如果其中一项有误，发出警告并断开连接，这一步保证了B的公钥的真实性。

产生一份秘密消息，这份秘密消息处理后将用作加密密钥，加密初始化向量（IV）和hmac的密钥。将这份秘密消息（协议中称为 per\_master\_secret）用B的公钥加密，封装成称作ClientKeyExchange的消息。由于用了B的公钥，保证了第三方无法窃听。

“我生成了一份秘密消息，并用你的公钥加密了，给你。”把ClientKeyExchange发给B。“注意，下面我就要用加密的办法给你发消息了！”

将秘密消息进行处理，生成加密密钥，加密初始化向量和hmac的密钥。

[我说完了。]

* B：

用自己的私钥将ClientKeyExchange中的秘密消息解密出来，然后将秘密消息进行处理，生成加密密钥，加密初始化向量和hmac的密钥，这时双方已经安全的协商出一套加密办法了。

“注意，我也要开始用加密的办法给你发消息了！”

[我说完了]

* A:

[我的秘密是...]

* B:

[其它人不会听到的...]

以上信息是否解决您的问题？