<https://mp.weixin.qq.com/s/UiGEzXoCn3F66NRz_T9crA>

# HTTPS实战之单向验证和双向验证

前面的文章中，提到了，https是在TCP协议与http之间加了一个控制安全传输的SSL协议，也就是说，直接运行在TCP之上的HTTP是普通的HTTP，运行在SSL/TLS上的HTTP则是HTTPS。这几个协议在计算机网络的OSI七层模型中的位置如下表所示：

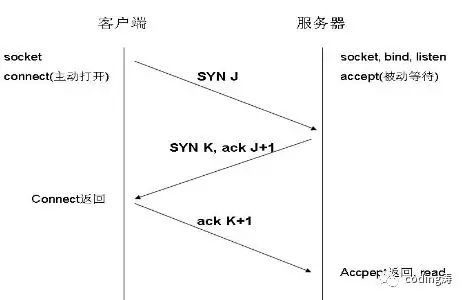
|  |  |  |
| --- | --- | --- |
| 层级 | 层名 | 常用协议 |
| 7 | 应用层 | **HTTP/HTTPS**、FTP、Socket、Telnet、SSH、SMTP、POP3、DHCP、DNS、NFS、SNMP |
| 6 | 表示层 | XDR、LPP |
| 5 | 会话层 | **SSL/TLS**、LDAP/DAP、RPC |
| 4 | 传输层 | **TCP**、UDP |
| 3 | 网络层 | IP、OSPF、ICMP、 |
| 2 | 数据链路层 | 以太网、令牌环、PPP、PPTP、L2TP、ARP、ATMP |
| 1 | 物理层 | 物理线路、光纤、无线电 |

客户端执行https请求时，需要由TCP协议建立和释放连接。这就涉及TCP协议的三次握手和四次挥手。

注意：我们可能会经常看到“HTTP的三次握手”这个说法，其实这种说法不准确，HTTP是没有什么三次握手的，这个其实指的就是TCP的三次握手。

## **一、TCP的三次握手（不想了解，可跳过）**

TCP通过三次握手，建立连接



第一次：客户端发送一个唯一的数据包(SYNJ)给服务器，请求建立连接

第二次：服务器收到客户端的请求后，生成一个SYN J的回应包(ack J+1)和一个新的数据包（SYN K），发给客户端

第三次：客户端收到服务器返回的两个包后，针对服务器的SYN K包生成一个回应包（ack K+1）,并发送给服务器。至此，完成3步握手。这个过程，类似我们在信号不好的时候打电话的过程：

“喂，能听到我说话吗”

“我能听到，你能听到我吗”

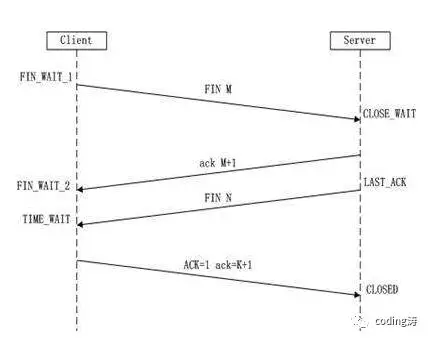
“能听到”

...

注意，这里**“能听到**”包含了**“听到”**和**“听懂”**的两层意思。想象一下，你用中文，对方给你回了一个你听不懂的俄文，这个通话也就没有必要进行下去了。

## **二、TCP的四次挥手（不想了解，可跳过）**

TCP协议是一个全双工协议（可以同时进行收发），通过四次挥手关闭连接。



简单理解：

第一步：客户端发送FIN（M）报文给服务器，告诉对方，“我的数据发完了”。

第二步：服务器收到FIN（M）报文后，回给客户端一个ack（M+1）报文，告诉客户端，“好，我知道了”。

第三步：服务器发一个FIN（N）报文给客户端，告诉对方，“我的数据也发完了”。

第四步：客户端回应ack（N+1）,告诉服务器，“好，我知道了”，至此，连接结束。

这个过程类似两个人打完电话的结束过程。

A:我说完了

B:哦

B想了一下，也没什么要说的

B:我也说完了

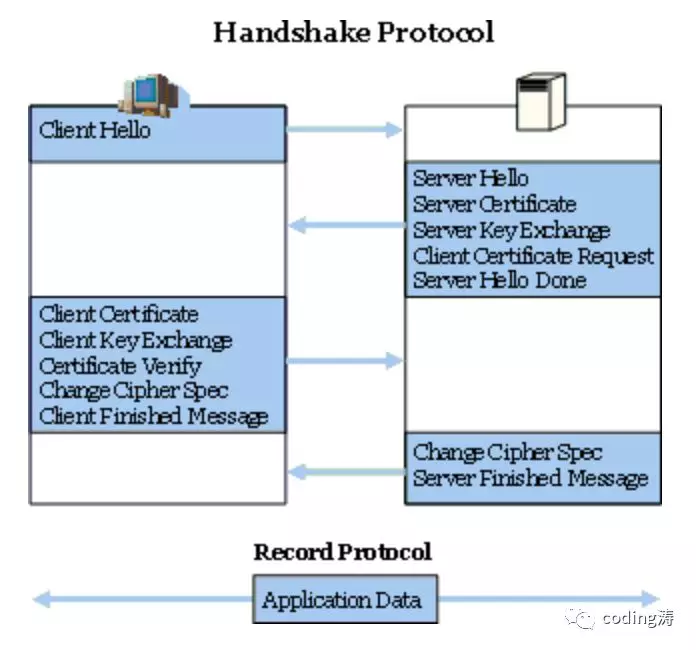
A:好

通话结束，双方挂电话

## **三、HTTPS单向验证**

TCP连接建立好后，对于HTTP而言，服务器就可以发数据给客户端。但是对于HTTPS，它还要运行SSL/TLS协议，SSL/TLS协议分两层，第一层是记录协议，主要用于传输数据的加密压缩；第二层是握手协议，它建立在第一层协议之上，主要用于数据传输前的双方身份认证、协商加密算法、交换密钥。

HTTPS验证过程就是SSL握手协议的交互过程。“HTTPS验证”这个说法其实不准确的，应该是“SSL验证”，这两种说法网上都能看到。



我们先从简单点的单向验证开始解释：

**第一步：客户端发起ClientHello**

客户端向指定域名的服务器发起https请求，请求内容包括:

1）客户端支持的SSL/TLS协议版本列表

2）支持的对称加密算法列表

3）客户端生成的随机数A

**第二步：服务端回应SeverHello**

服务器收到请求后，回应客户端，回应的内容主要有：

1）SSL/TLS版本。服务器会在客户端支持的协议和服务器自己支持的协议中，选择双方都支持的SSL/TLS的最高版本，作为双方使用的SSL/TLS版本。如果客户端的SSL/TLS版本服务器都不支持，则不允许访问

2）与1类似，选择双方都支持的最安全的加密算法。

3）从服务器密钥库中取出的证书

4）服务器端生成的随机数B

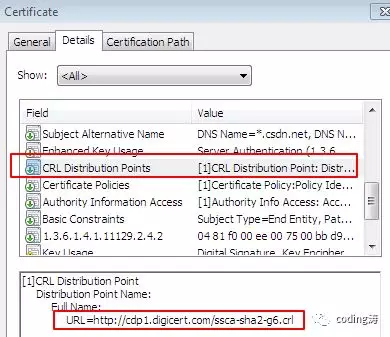
**第三步：客户端回应**

客户端收到后，检查证书是否合法，主要检查下面4点：

**1、检查证书是否过期**

**2、检查证书是否已经被吊销**。

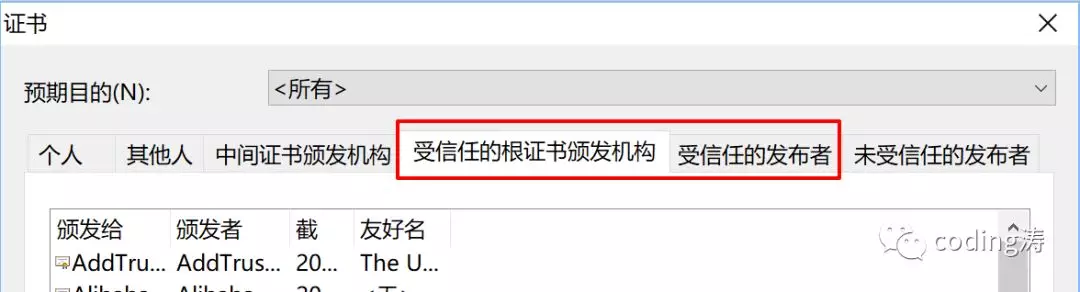
有CRL和OCSP两种检查方法。CRL即证书吊销列表，证书的属性里面会有一个CRL分发点属性，如下图所示（CSDN的证书），这个属性会包含了一个url地址，证书的签发机构会将被吊销的证书列表展现在这个url地址中；OCSP是在线证书状态检查协议，客户端直接向证书签发机构发起查询请求以确认该证书是否有效。



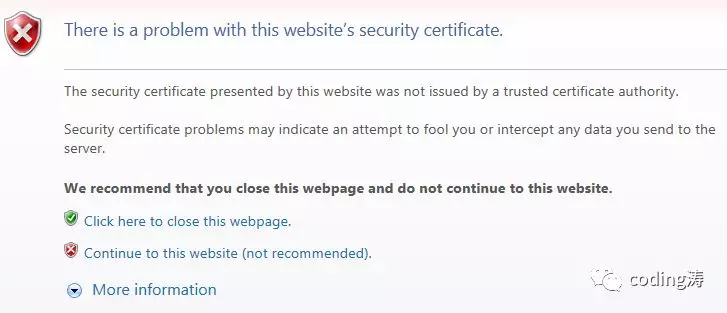
**3、证书是否可信**。

客户端会有一个信任库，里面保存了该客户端信任的CA（证书签发机构）的证书，如果收到的证书签发机构不在信任库中，则客户端会提示用户证书不可信。

* 若客户端是浏览器，各个浏览器都会内置一些可信任的证书签发机构列表，在浏览器的设置中可以看到。



如果不在信任表中，则浏览器会出现类似下面的警告页面，提示你不安全。（当然，你可以选择继续访问）



* 若客户端是程序，例如Java中，需要程序配置信任库文件，以判断证书是否可信，如果没设置，则默认使用jdk自带的证书库（jre\lib\security\cacerts,默认密码changeit）。如果证书或签发机构的证书不在信任库中，则认为不安全，程序会报错。（你可以在程序中设置信任所有证书，不过这样并不安全）。

**4、检查收到的证书中的域名与请求的域名是否一致。**

**若客户端是程序，这一项可配置不检查。若为浏览器，则会出现警告，用户也可以跳过。**

证书验证通过后，客户端使用特定的方法又生成一个随机数c，这个随机数有专门的名称“pre-master key”。接着，客户端会用证书的公钥对“pre-master key”加密，然后发给服务器。

**第四步，服务器的最后回应**

服务器使用密钥库中的私钥解密后，得到这个随机数c。此时，服务端和客户端都拿到了随机数a、b、c，双方通过这3个随机数使用相同的**DH密钥交换算法**计算得到了相同的**对称加密的密钥**。这个密钥就作为后续数据传输时对称加密使用的密钥。

服务器回应客户端，握手结束，可以采用对称加密传输数据了。

这里注意几点：

1、整个验证过程，折腾了半天，其实是为了安全地得到一个双方约定的对称加密密钥，当然，过程中也涉及一些身份认证过程。既然刚开始时，客户端已经拿到了证书，里面包含了非对称加密的公钥，为什么不直接使用非对称加密方案呢，这是因为非对称加密计算量大、比较耗时，而对称加密耗时少。

2、对称加密的密钥只在这次连接中断前有效，从而保证数据传输安全。

3、为什么要用到3个随机数，1个不行吗？这是因为客户端和服务端都不能保证自己的随机数是真正随机生成的，这样会导致数据传输使用的密钥就不是随机的，时间长了，就很容易被破解。如果使用客户端随机数、服务端随机数、pre-master key随机数这3个组合，就能十分接近随机。

4、什么是信任库和密钥库。信任库前面已经说了，它是用来存放客户端信任的CA的证书。在程序交互中，需要确保你访问的服务器的证书在你的信任库里面。密钥库是用来存放服务器的私钥和证书。

5、中间人攻击问题。前面过程说明中，有一点，客户端是验证有问题的时候，是可以选择继续的。对浏览器而言，用户可以选择继续访问；对程序而言，有些系统为了处理简单，会选择信任所有证书，这样就给中间人攻击提供了漏洞。

中间人攻击时，它想办法拦截到客户端与服务器之间的通信。在客户端向服务器发信息时，中间人首先伪装成客户端，向真正的服务器发消息，获得真正的证书，接着伪装成服务器将自己的伪证书发给客户端。服务器向客户端发消息时，中间人伪装成客户端，接收消息，然后再伪装成服务器向客户端发消息。最后验证过程完成后，客户端的真实对称密钥被中间人拿到，而真正的服务器拿到的是中间人提供的伪密钥。后续数据传输过程中的数据就会被中间人窃取。

## **四、HTTPS双向验证**

单向验证过程中，客户端会验证自己访问的服务器，服务器对来访的客户端身份不做任何限制。如果服务器需要限制客户端的身份，则可以选择开启服务端验证，这就是双向验证。从这个过程中我们不难发现，**使用单向验证还是双向验证，是服务器决定的。**

一般而言，我们的服务器都是对所有客户端开放的，所以**服务器默认都是使用单向验证**。如果你使用的是Tomcat服务器，在配置文件server.xml中，配置Connector节点的clientAuth属性即可。若为true，则使用双向验证，若为false，则使用单向验证。如果你的服务，只允许特定的客户端访问，那就需要使用双向验证了。

双向验证基本过程与单向验证相同，不同在于：

1）第二步服务器第一次回应客户端的**SeverHello**消息中，会要求客户端提供“客户端的证书”

2）第三步客户端验证完服务器证书后的回应内容中，会增加两个信息：

1、客户端的证书

2、客户端证书验证消息（**CertificateVerify message**）：客户端将之前所有收到的和发送的消息组合起来，并用hash算法得到一个hash值，然后用客户端**密钥库**的**私钥**对这个hash进行签名，这个签名就是**CertificateVerify message**

说明：这里关于客户端私钥的使用，网上有很多文章认为：在协商对称加密方案时，服务端先用客户端公钥加密服务器选定的对称加密方案，客户端收到后使用私钥解密得到。首先，对称加密方案就那么几种，逐个试试就能试出来，没必要为了这个增加一个客户端和服务端的交互过程。而这里关于**CertificateVerify message**的说法参考了维基百科关于“Transport Layer Security”一文中"Client-authenticated TLS handshake"的描述。链接：https://en.wikipedia.org/wiki/Transport\_Layer\_Security#Client-authenticated\_TLS\_handshake

3）服务器收到客户端证书后：

a）确认这个证书是否在自己的**信任库**中（当然也会校验是否过期等信息），如果验证不通过则会拒绝连接；

b）用**客户端证书中的公钥**去验证收到的证书验证消息中的签名。这一步的作用是为了确认证书确实是客户端的。

所以，**在双向验证中，客户端需要用到密钥库，保存自己的私钥和证书，并且证书需要提前发给服务器，由服务器放到它的信任库中。**

**五、总结一下：**

1、单向验证中，如果是你客户端，你需要拿到服务器的证书，并放到你的信任库中；如果是服务端，你要生成私钥和证书，并将这两个放到你的密钥库中，并且将证书发给所有客户端。

2、双向验证中，如果你是客户端，你要生成客户端的私钥和证书，将它们放到密钥库中，并将证书发给服务端，同时，在信任库中导入服务端的证书。如果你是服务端，除了在密钥库中保存服务器的私钥和证书，还要在信任库中导入客户端的证书。

3、再次强调，使用单向验证还是双向验证，是服务器决定的。

4、https的验证过程，不管是单向还是双向，只有**四步**，网上很多关于https验证过程的文章中，写了来来回回七八上十步。要真是这样，访问一个https地址，时间全花在了交互上了。