文 TLS总结（下）——TLS如何保证安全

网络安全 网络传输协议 tls **jimsshom** 1 天前发布

上篇主要是介绍了HTTP存在的两大安全问题

- 明文
- 无法验证服务器的真实性

从而引出了TLS。本篇就来着重介绍下TLS。

说起TLS可能有些人还比较陌生，但如果说到SSL，那知道的人就更多了。TLS其实就是SSL发展而来，版本演进大体为SSL 2.0 -> SSL 3.0 -> TLS 1.0（可以看做是SSL 3.1版）。

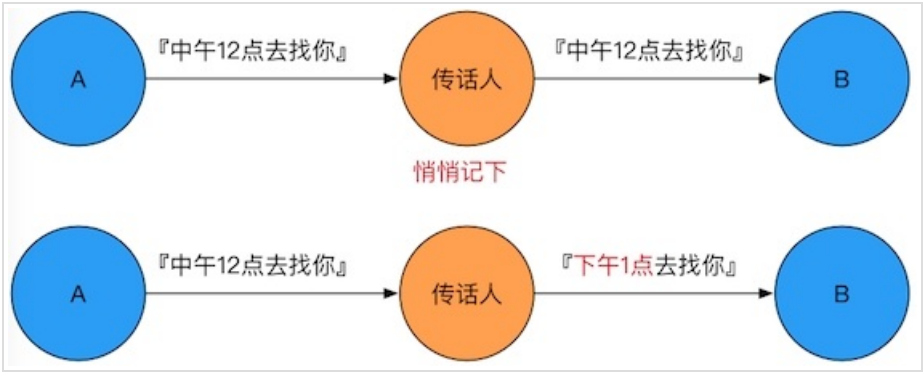
TLS主要提供三个基本服务

- 加密
- 身份验证
- 消息完整性校验

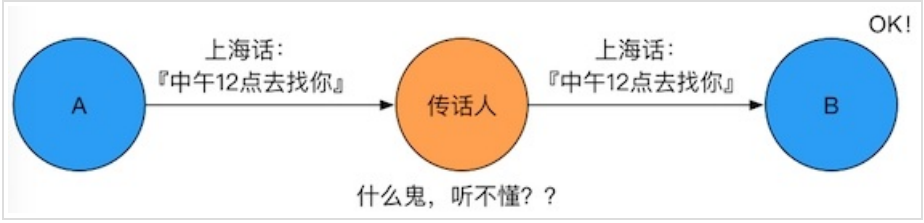
其中第三点是网络协议中常用的一个校验和机制，和我们这边要说的安全话题不是太相关，不再赘述。而前两点正好是对应了HTTP的两个问题，下面分别介绍下。

内容加密

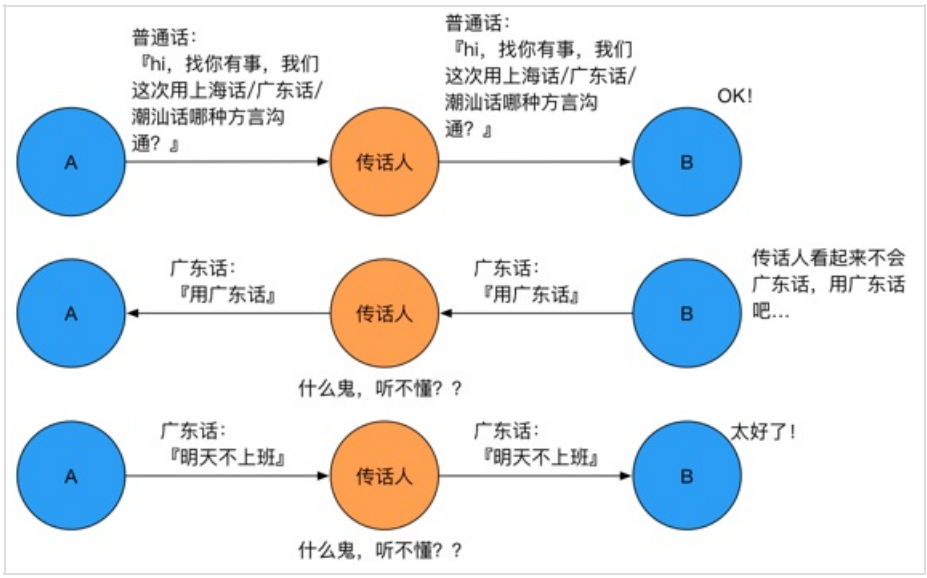
其实网络通信可以类比为人之间的对话，比如A和B之间需要沟通，但两人没法直接见面，但有一个传话人可以代为传话。这样两人就能间接沟通了，只不过这个传话的过程是有一定风险（被记录，被篡改）的。



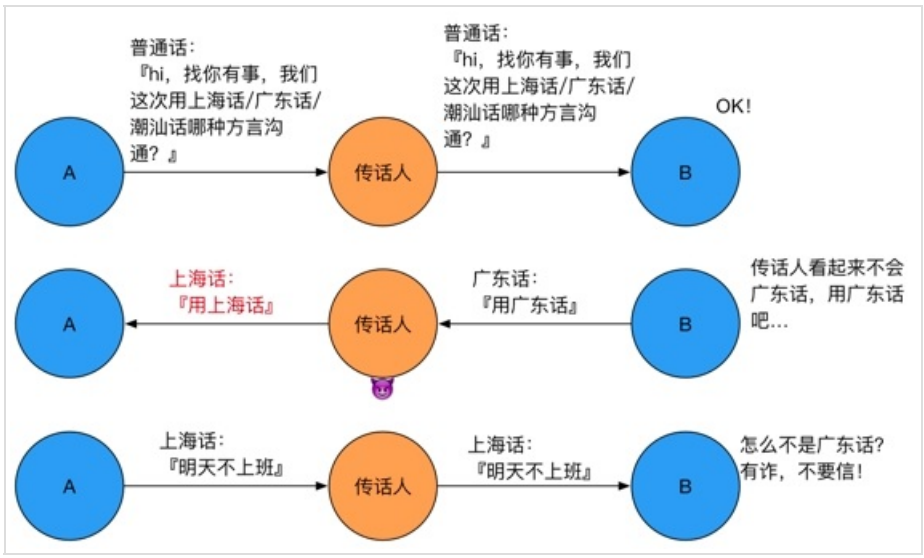
为了避免这种风险，久而久之A和B想出了一个办法：他们约定使用传话人不会的上海话沟通。这样每次传话人都听不懂啥意思，只能尽可能有样学样地把A说给他的话学给B听。



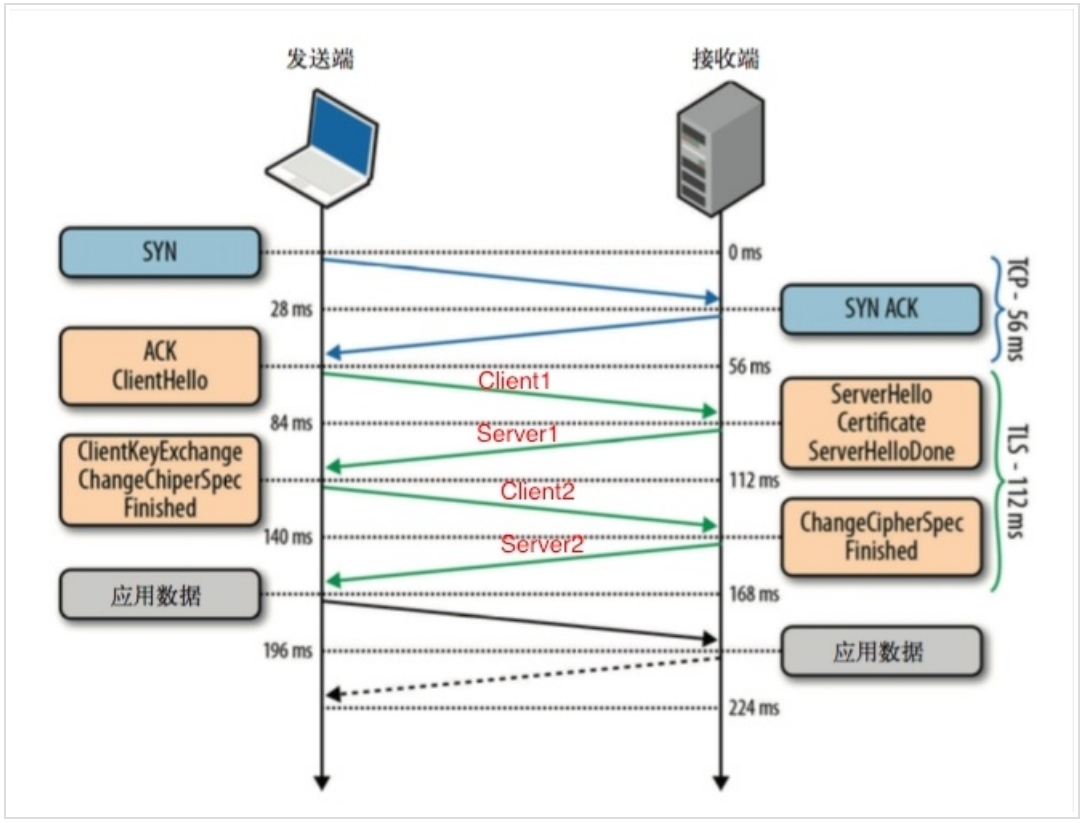
这个普通话翻译成上海话的过程，对于传话人来说就是一种内容的『加密』。但这个方法不一定是唯一的，可能以后另一个传话人懂上海话呢？于是在每次使用方言沟通之前，A和B会先行沟通下他们这次对话使用的语种（两人真是牛逼！）



这个选择语种的过程，其实就是两人之间关于翻译方式的『协商』过程。传话人要想从中捣鬼，就能立马被发现，这样AB两人为了沟通安全就会停止这次沟通，转而找其他的传话人传话了。



以上看到AB两人机智的采用『翻译』『协商』两招完成的安全沟通了。将同样的方法拿到网络通信中，就是TLS的加密机制了。



这个图描述了发送端和接收端是怎么协商他们之间的加密机制的。TLS协议是基于TCP协议之上的，图中第一个蓝色往返是TCP的握手过程，不在本文范畴中按下不表。之后的两次橙色的往返，就是我们要重点说的协商过程了，可以叫做TLS的握手。握手过程如下：

1. Client1：TLS版本号 + 所支持加密套件列表 + 希望使用的TLS选项
2. Server1：选择一个客户端的加密套件 + 自己的公钥 + 自己的证书 + 希望使用的TLS选项 +（要求客户端证书）
3. Client2：（自己的证书）+ 使用服务器公钥和协商的加密套件加密一个对称密钥
4. Server2：使用私钥解密出对称密钥后，发送的加密的Finish消息，表明完成握手

我们一步一步来仔细看。

第一步，客户说明他支持的所有加密套件（就像上海话、广东话一样）让服务端选择一个。这里和Server1还完成了一个TLS选项的协商，这可以理解为在确定核心的加密方式的同时，顺便沟通下是否能支持一些『附加功能』。这些附加功能先按下不表，后续会介绍两个常用的。

第二步，服务端选择了加密套件。加密算法都是需要一个秘钥的，服务端这里给出的是自己的公钥。这里的公钥其实蕴含着一个关键点：秘钥协商过程采用了不对称加密。简单来说，一般的对称加密如下

> encrypt(明文，秘钥) = 密文
>
> decrypt(密文，秘钥) = 明文

也就是说加密和解密用的是一个秘钥。而非对称加密就比较神奇了

> encrypt(明文，公钥) = 密文
>
> decrypt(密文，私钥) = 明文

加密和解密是需要不同的密钥的。在TLS握手前，客户端本身不知道服务器的秘钥的，如果运用对称加密，那么服务端告诉客户端秘钥的过程中，两者之间的中间节点其实也拿到秘钥了，这样后续两者之间的加密通信，对于中间节点来说也能解密出内容了。因此这里采用不对称加密，服务端给出一个公钥供客户端后续使用。此外这一步服务端还提供了证书，并且可能要求客户端提供证书。关于证书下文会提到，只要有了证书，就能保证和你通信的对方是真实的，而不是别人伪造的。这里只要先理解，客户端看到服务端证书后，就能够相信服务端并使用服务端给的公钥了。

第三步，客户端给出了双方后续加密通信所要使用的对称密钥，通过服务器公钥加密后返回给服务端。由于对称密钥是被不对称加密算法加密起来的，因此中间节点拿到后，没有办法解密出真正的内容。

第四步，服务端拿公钥对应的私钥解密出真正的对称密钥，至此加密算法和加密所需的密钥都确定，服务端给客户端发送一个finish完成握手，证明双方都已经准备好加密通信了。之后双方的交互都可以使用对称加密算法加密了。

TLS扩展

说完了如何确定加密的算法和密钥后，我们再说说那所谓的『附加功能』。所谓TLS的扩展，就是在TLS协议核心保持不变的基础上，对其进行扩展以使其支持更多的特性，这里简单介绍下两个扩展：ALPN和SNI。

ALPN（Application Layer Protocol Negotiation）即应用层协议协商，旨在将原来应用层协议中的协议协商，提前合并到TLS握手时完成，减少一次往返时间。这里通过一个例子来说明下什么叫做应用层的协议协商。假设TLS握手完成了，接下来客户端和服务端就应该开始进行正常的HTTP通信了，但是客户端认为HTTP协议比较慢（还记得上篇提到的流式传输的效率问题吗），在请求内容之前先向服务端提出『我们能不能换HTTP2.0通信』。服务端会根据自己支持HTTP2.0的情况给出答复，之后客户端会根据服务端的答复，使用HTTP或HTTP2.0向服务端请求内容。可以看到这个过程中又多了一次协商升级协议的往返，导致客户端拿到返回结果的延迟更大了。为了优化这一点，在HTTP使用TLS加密的时候，会使用ALPN扩展，把协商升级协议的请求放在Client1中，服务端在Server1就返回说是否可以升级协议。这样TLS握手之后，客户端就可以直接按HTTP2.0发起请求而不必多一次往返了。

SNI（Server Name Indication）叫做服务器名称指示，主要是支持服务端同一个机器部署有多个站点的情况的。前面说的证书、公钥私钥，都是针对某个站点来说的。网站A和网站B可能租了同一台服务器，但他们的证书和密钥一定是不同的。当客户端请求到这台公用服务器时，服务器也不知道该返回哪个站点的信息了。这个时候使用SNI扩展，在client1带上访问站点的信息，服务端看到后，就会去找对应站点的证书、密钥了。

TLS的代价

任何事物都是有利有弊，引入TLS机制固然是能够保证安全，但却在TCP握手和HTTP通信之间，多加了两个往返的TLS握手过程。特别是现代网页应用中充满了AJAX请求的场景下，请求的往往都是一个很小的资源，在一波TCP包中就能返回的。这种情况下握手需要三次往返，而真正有意义的请求往返只有一次，安全的代价非常大！（关于延迟对于性能的重要性，怎么强调都不为过，但往往会被人所忽视。这里不展开，但可以仔细思考下TCP的拥塞避免策略：在大延迟的情况下TCP包需要等上一波TCP包的往返后才会继续发送。这就是为什么很多人都是50兆100兆的宽带还经常觉得上网慢的

原因，因为延迟大！而运营商是从来不会宣传或保证延迟这个指标的）

当然，还是有优化的办法的。上篇说过，现代web应用经常是打开一个网页同时会发送几十个请求。TLS握手这个协商加密套件和密钥的过程，需要重复几十次吗？当然不用。既然这几十个请求都是请求的同一个网站，那共用第一次请求的协商结果就好了。这样虽然第一次请求中TLS握手的代价很大（比如占了50%），但从整个网页全局看来，代价就降低到了百分之几了。目前一些主流浏览器的实现也确实利用了这个机制：他们会刻意先发起一个请求，待该请求的TLS链接完成后，再并发发起其他的请求。后续并发的请求就能够利用TLS协商后的结果了。

这种对协商结果的复用叫做TLS的『会话恢复机制』，主要有两种方案：Session Identifier 和 Session Ticket。

Session Identifier即会话标识符，是一种服务端保存会话信息的方案。

- 服务端为每个客户端保存会话ID和协商后的会话参数
- 会话ID通过Server1带给客户端
- 客户端在后续创建连接的Client1中带上会话ID
- 如果两方都还记得会话ID，则可开启『简短握手』，一次往返即可
- 如该会话ID的信息在服务端不存在，则继续正常握手的第二个往返

方案整体思路其实和web应用的session类似，都是在服务端维护信息，客户端提供一个sessionId。因此该方案面临和session一样的挑战：服务端需要管理海量会话信息并且制定适当的淘汰策略。

Session Ticket即会话记录单，是一种客户端保存会话信息的方案。

- 协商成功后会话信息由服务器通过其私钥加密后携带在最后一次交换中
- 由客户端保存这个加密后的会话信息，但其无法解密得知具体内容
- 下次连接时客户端将其带在Client1的SessionTicket扩展信息中

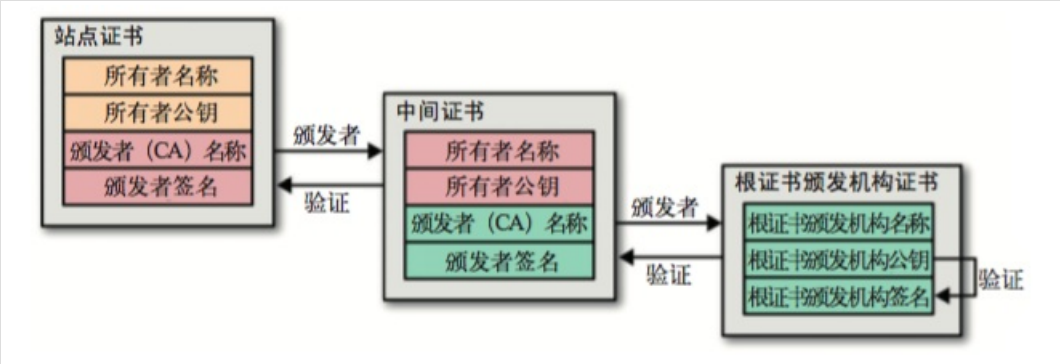
这个方案的整体思路就和cookie类似了，有效地减轻了服务端的压力。但这属于一种较新的机制，目前还不是所有客户端都支持。

证书机制

上面这么大篇幅都是介绍的TLS如何完成『加密』这个事情，然而之前说到的HTTP另一个问题『无法验证服务器的真实性』还是没有得到解决。毕竟如果有个节点要伪装成服务器，和用户进行TLS的加解密交互，也是完全没问题的吧。因此，TLS中还需要通过『证书机制』来保证你所访问的服务器是真实的。从安全层面来说，『证书』和『加密』是同等重要的两个机制。

做个假设，你和小A是朋友，你们之间什么事都可以畅所欲言。你们当面聊天时，你不介意把自己的隐私和小A分享。你的分享，其实是建立在『当面』的基础上的，也就是你通过『辨识小A的外貌』来断定他的真实性后，才会信任他与他分享。如果有一天小A通过一个陌生的电话联系你，那你恐怕就不会这么毫无防备了。为了验证电话那头真的是小A，你们之间可能需要一些验证机制，比如让小A说说你们小时候发生的某个事或者在你们两个面对面的时候确定个暗号。这个事件或者暗号，其实就是你用来认证小A的『证书』了。 这样你和小A之间的认证就OK了。可有一天，一个自称小A朋友的小B找到你，想叫你帮个忙，可以你之前从来没听说过小B。这个时候为了验证小B是否是可信的，你只能找来小A帮忙了。你联系小A使用你们之间的『证书』验证其真实性后，由小A使用他和小B约定的『证书』来验证小B的真实性。验证成功后，你也就能信任小B了吧，毕竟你的朋友小A已经信任电话那头的小B了。

这个找小A验证小B的过程，正是TLS中核心的『证书链』机制。



前面说到，在TLS握手过程中服务端会提供给客户端它的证书。这个证书可不是随意生成的，而是通过指定的权威机构申请颁发的。服务端如果能够提供一个合法的证书，说明这个服务端是合法的，可以被信任。就拿上图来说

1. 客户端获取到了站点证书，拿到了站点的公钥

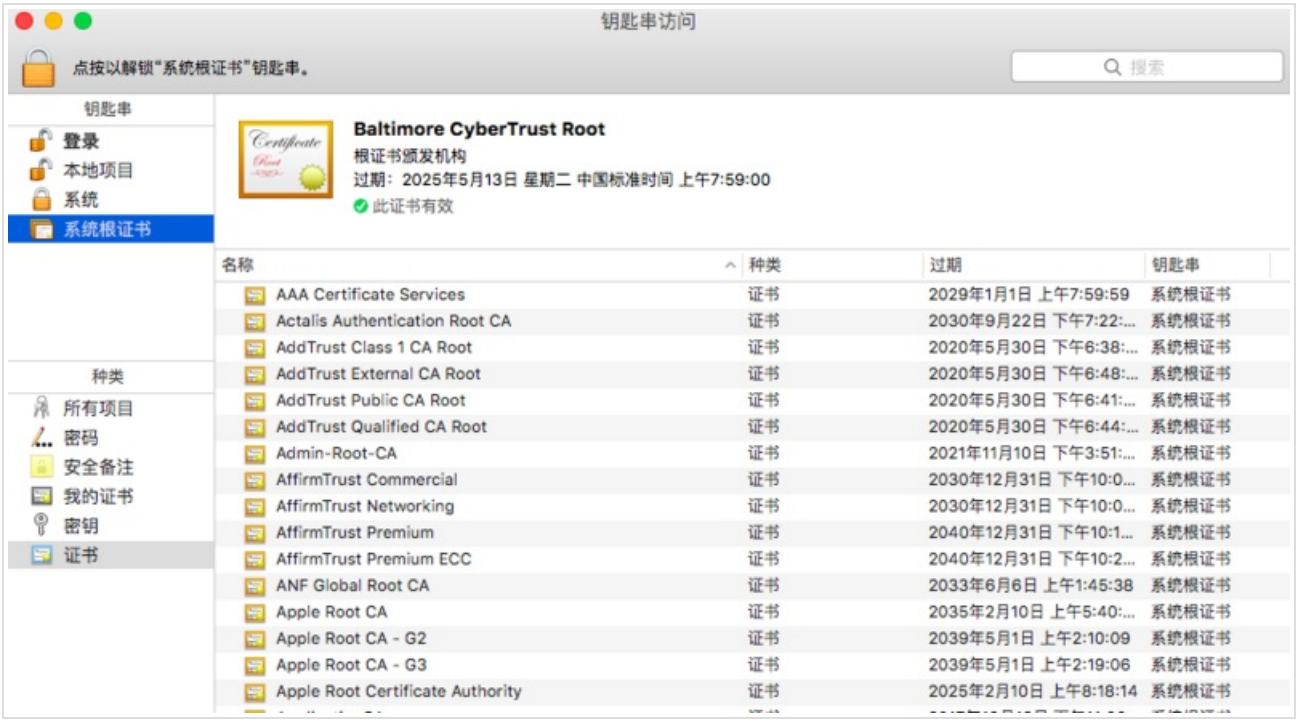
2. 要验证站点可信后，才能使用其公钥，因此客户端找到其站点证书颁发者的信息
3. 站点证书的颁发者验证了服务端站点是可信的，但客户端依然不清楚该颁发者是否可信
4. 再往上回溯，找到了认证了中间证书商的源头证书颁发者。由于源头的证书颁发者非常少，我们浏览器之前就认识了，因此可以认为根证书颁发者是可信的
5. 一路倒推，证书颁发者可信，那么它所颁发的所有站点也是可信的，最终确定我们所要访问的服务端是可信的
6. 客户端使用证书中包含的公钥，继续完成TLS的握手过程

整个过程包含两个关键点

- 根证书颁发机构的权威性需要保证
- 如何从证书颁发者那里验证证书的合法性

先说说第一个问题，权威的根证书颁发机构非常少，因此浏览器和操作系统都会内置一些可信的根证书颁发机构，也就是说这些机构的权威性是由浏览器或操作系统保证的。但曾经也出现过一些并不权威的证书由于某些目的被内置在中国版的Firefox中这样的事件，让人直呼防不胜防。

好在当前大多数情况下跟证书还是能够保证权威的，如果是CNNIC这种系统性风险，那也只有靠多关注安全新闻来及时避免了。通过系统或浏览器配置就能查看到自己内置可信的跟证书颁发机构了，这里是我mac的结果。如果发生了像上述CNNIC证书的问题，只要到证书列表这里删除对应的可信证书即可。



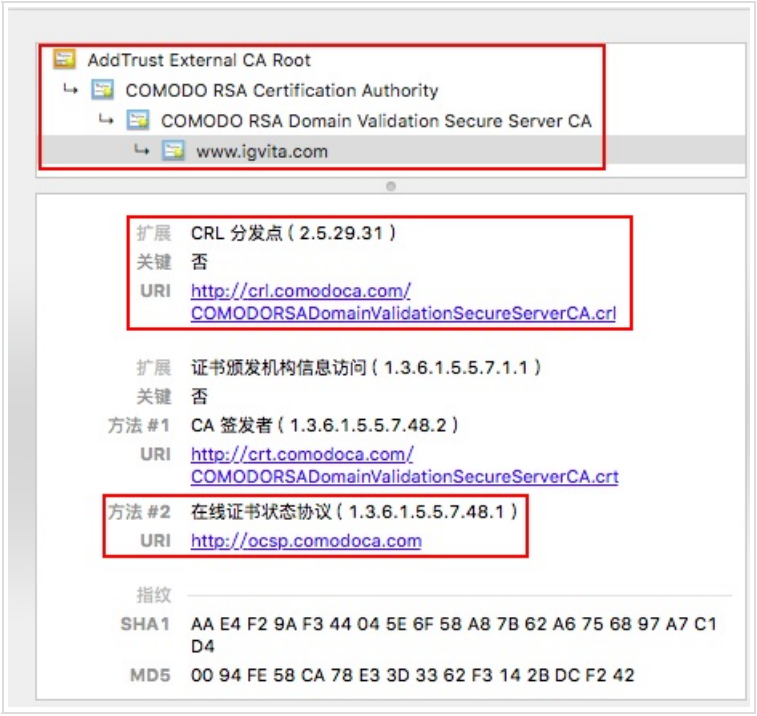
再来说说第二个问题，如何从证书颁发者那里验证某个证书的有效性。每个证书颁发机构在颁发证书的同时，有义务维护其认证站点的权威性：比如每

个证书都有过期时间，一旦到期就要立即失效；某个站点需要更换证书服务商，换上新的证书时，老的也就立即要失效，否则老的证书可能会被挪用到不法站点上；由于站点自身问题导致私钥泄密，那么TLS的加密就不能保证安全；证书颁发机构被冒名顶替等等。一旦发生以上任何一种情况，证书颁发者都要保证对应证书立即失效。证书颁发者一般提供两种方式来验证证书有效性：CRL和OCSP。

CRL（Certificate Revocation List）即证书撤销名单。证书颁发者会提供一份已失效证书的名单，供浏览器验证证书使用。当然这份名单是巨长无比的，浏览器不可能每次TLS都去下载，所以常用做法是浏览器会缓存这份名单，定期做后台更新。这样虽然后台更新存在时间间隔，证书失效不实时，但一般也OK了。

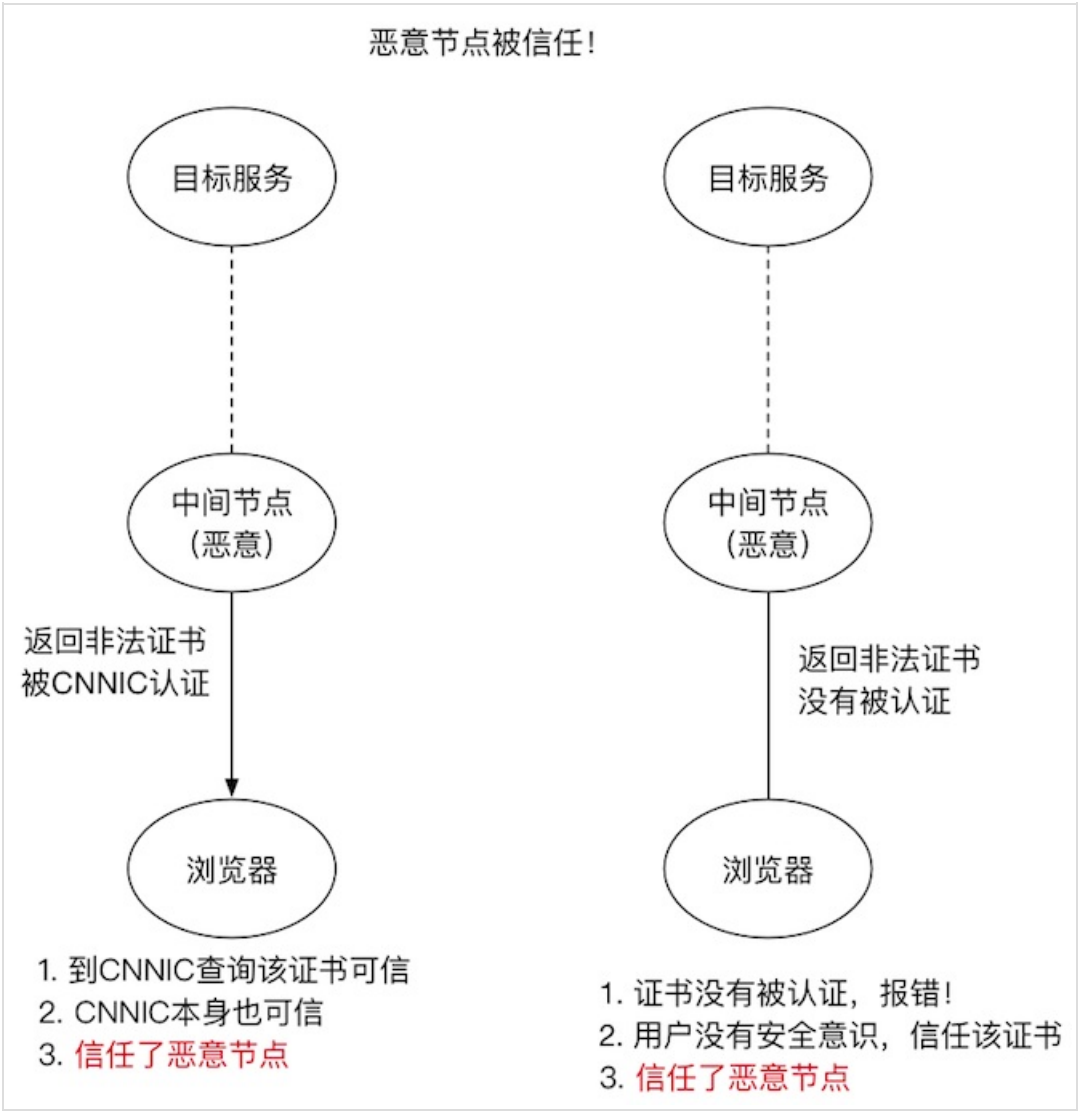
OCSP（Online Certificate StatusProtocol）即在线证书状态协议。除了离线文件，证书颁发者也会提供实时的查询接口，查询某个特定证书目前是否有效。实时查询的问题在于浏览器需要等待这个查询结束才能继续TLS握手，延迟会很大。此外，通过『某个IP请求了某个证书的OCSP请求』这个信息，其实就暴露了某个用户正在访问某个网站了，也算是一种隐私的泄露。

以上是站在证书颁发者的角度说明会提供两种判断方式，实际情况下浏览器究竟会选择哪种方式判断，每个浏览器都有自己的实现。下图是通过chrome查看某个网站的证书信息，可以看到其中的证书链和证书验证信息。



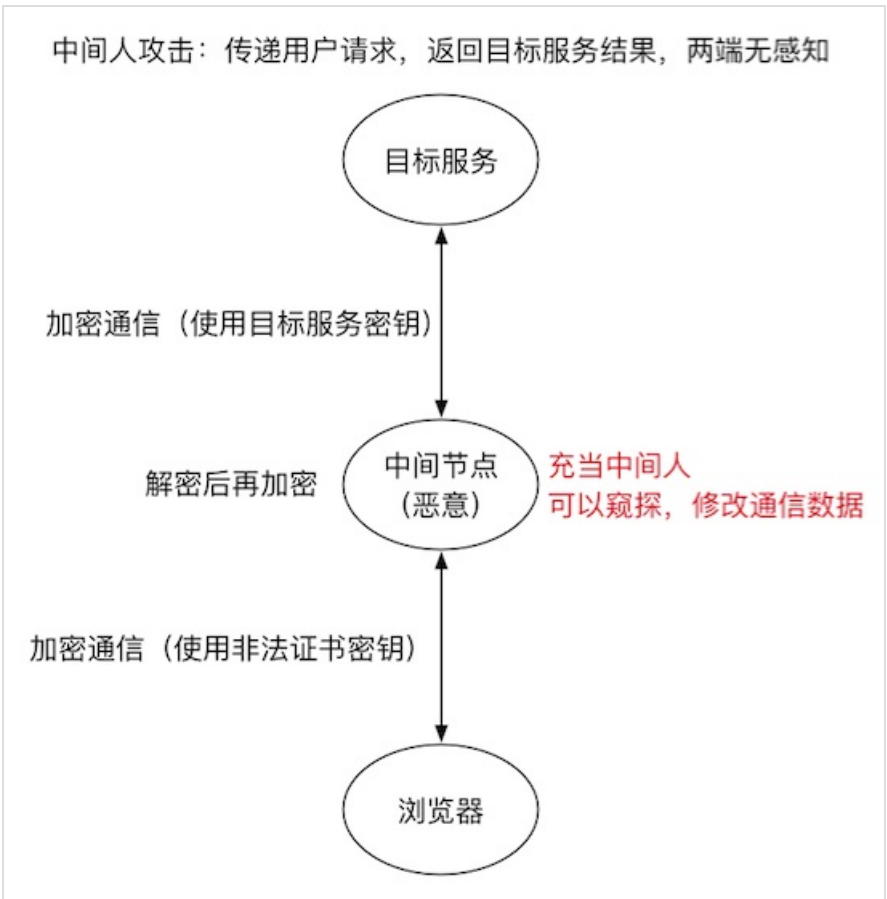
中间人攻击

介绍完证书如何保证站点的可信以后，我们来看看如果证书不合法会怎么样。导致不合法证书还能被使用有两种情况，一种是前文提到的CNNIC证书等类似的系统性风险，另一种是用户缺乏安全意识，手动信任了存在安全隐患的证书。



对于前者，我们要多关注安全新闻定期排雷，对于后者我们要提高安全意识，以后看到这些画面时可要慎之又慎了，而不是一味地点同意。

一旦非法的证书被浏览器视为合法，那么就面临着一种『中间人攻击』的风险：恶意节点会通过持有的非法证书和客户端交互，之后代替客户端，向真实服务器发起请求，并把服务器的请求返回给客户端。这样客户端好像感觉自己直接连接了服务器，但实际上有个隐藏的中间人，神不知鬼不觉地窃取了中间的信息。



至此，已经将TLS技术如何应对『明文』和『无法验证服务器的真实性』两个主要安全问题讲完了，由于本人并不从事网工相关职业，因此就没有深究到协议消息帧细节，只是从原理层面进行了理解。如果关于TLS还有什么没有提到，或者理解不正确的，欢迎各位提出来~

1 推荐 收藏

你可能感兴趣的文章

GitCafe已正式取消对 SSLv3 协议的支持，目前选择支持更安全可靠的 TLS 协议 2k 浏览

白话解释 OSI模型，TLS/SSL 及 HTTPS 12 收藏，1.3k 浏览

TLS总结（上）——我们为啥需要TLS 5 收藏，368 浏览

讨论区

使用评论询问更多信息或提出修改意见，请不要在评论里回答问题

提交评论

评论支持部分 Markdown 语法：`**bold**` `_italic_` `[link](http://example.com)` `> 引用` `` `code` `` `- 列表`。
同时，被你 @ 的用户也会收到通知

本文隶属于专栏

jimsshom的总结分享

个人日常工作生活的所用所学进行总结的地方

jimsshom
作者

关注专栏

系列文章

TLS总结（上）——我们为啥需要TLS 5 收藏， 368 浏览

相关收藏夹

换一组

http
4 个条目 | 0 人关注

http
5 个条目 | 0 人关注

linux
5 个条目 | 0 人关注

分享扩散：