20 张图彻底弄懂 HTTPS 的原理！

ImportNew 12月10日

以下文章来源于码海 ，作者码海

码海

一起进阶，一起牛逼！这里将会分享计算机底层、计算机网络、操作系统，Java、框架...

微信扫一扫
关注该公众号

(给ImportNew加星标，提高Java技能)

> 转自：码海

前言

近年来各大公司对信息安全传输越来越重视，也逐步把网站升级到 HTTPS 了，那么大家知道 HTTPS 的原理是怎样的吗，到底是它是如何确保信息安全传输的？网上挺多介绍 HTTPS，但我发现总是或多或少有些点有些遗漏，没有讲全，今天试图由浅入深地把 HTTPS 讲明白，相信大家看完一定能掌握 HTTPS 的原理，本文大纲如下：

1. HTTP 为什么不安全
2. 安全通信的四大原则
3. HTTPS 通信原理简述
   - 对称加密
   - 数字证书
   - 非对称加密
   - 数字签名
4. 其它 HTTPS 相关问题

HTTP 为什么不安全

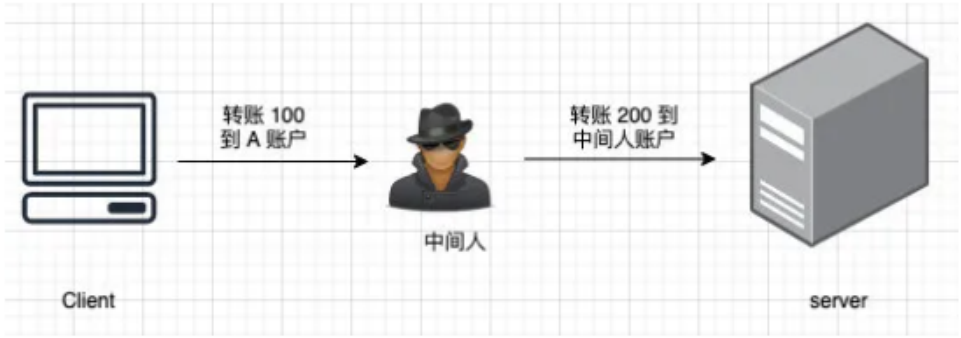
HTTP 由于是明文传输，主要存在三大风险

1、 窃听风险

中间人可以获取到通信内容，由于内容是明文，所以获取明文后有安全风险

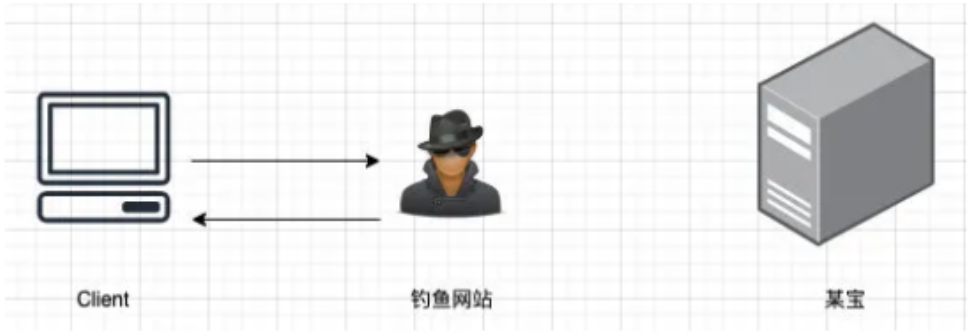
2、 篡改风险

中间人可以篡改报文内容后再发送给对方，风险极大



3、 冒充风险

比如你以为是在和某宝通信，但实际上是在和一个钓鱼网站通信。



HTTPS 显然是为了解决这三大风险而存在的，接下来我们看看 HTTPS 到底解决了什么问题。

安全通信的四大原则

看了上一节，不难猜到 HTTPS 就是为了解决上述三个风险而生的，一般我们认为安全的通信需要包括以下四个原则：**机密性**、**完整性**，**身份认证**和**不可否认**

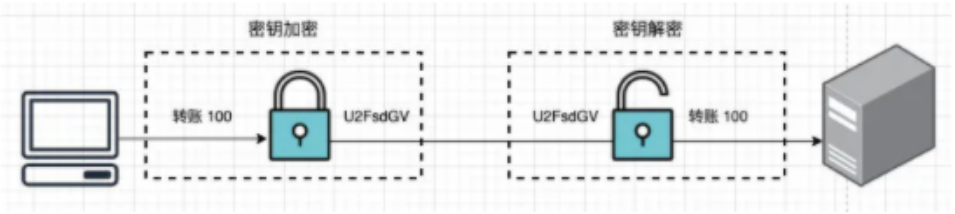
1. **机密性**：即对数据加密，解决了窃听风险，因为即使被中间人窃听，由于数据是加密的，他也拿不到明文
2. **完整性**：指数据在传输过程中没有被篡改，不多不少，保持原样，中途如果哪怕改了一个标点符号，接收方也能识别出来，从来判定接收报文不合法
3. **身份认证**：确认对方的真实身份，即证明「你妈是你妈」的问题，这样就解决了冒充风险，用户不用担心访问的是某宝结果却在和钓鱼网站通信的问题
4. **不可否认**：即不可否认已发生的行为，比如小明向小红借了 1000 元，但没打借条，或者打了借条但没有**签名**，就会造成小红的资金损失

接下来我们一步步来看看 HTTPS 是如何实现以满足以上四大安全通信原则的。

HTTPS 通信原理简述

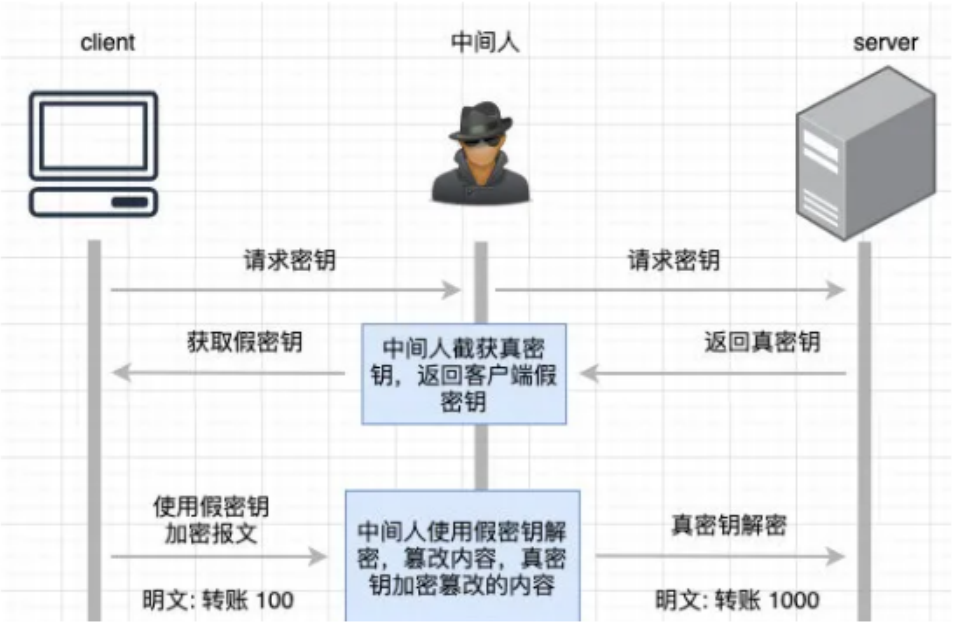
对称加密：HTTPS 的最终加密形式

既然 HTTP 是明文传输的，那我们给报文加密不就行了，既然要加密，我们肯定需要通信双方协商好密钥吧，一种是通信双方使用**同一把密钥**，即**对称加密**的方式来给报文进行加解密。



如图示：使用对称加密的通信双方使用**同一把**密钥进行加解密。

对称加密具有加解密速度快，性能高的特点，也是 HTTPS 最终采用的加密形式，但是这里有一个关键问题，对称加密的通信双方要使用同一把密钥，这个密钥是如何协商出来的？如果通过报文的方式直接传输密钥，之后的通信其实还是在裸奔，因为这个密钥会被中间人截获甚至替换掉，这样中间人就可以用截获的密钥解密报文，甚至替换掉密钥以达到篡改报文的目的。



有人说对这个密钥加密不就完了，但对方如果要解密这个密钥还是要传加密密钥给对方，依然还是会被中间人截获的，这么看来直接传输密钥无论怎样都无法摆脱俄罗斯套娃的难题，是不可行的。

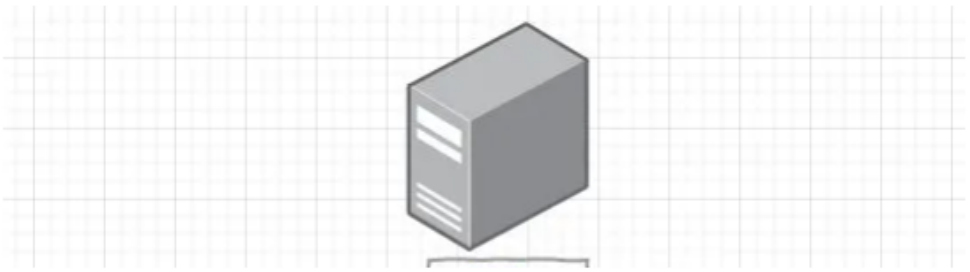
非对称加密：解决单向对称密钥的传输问题

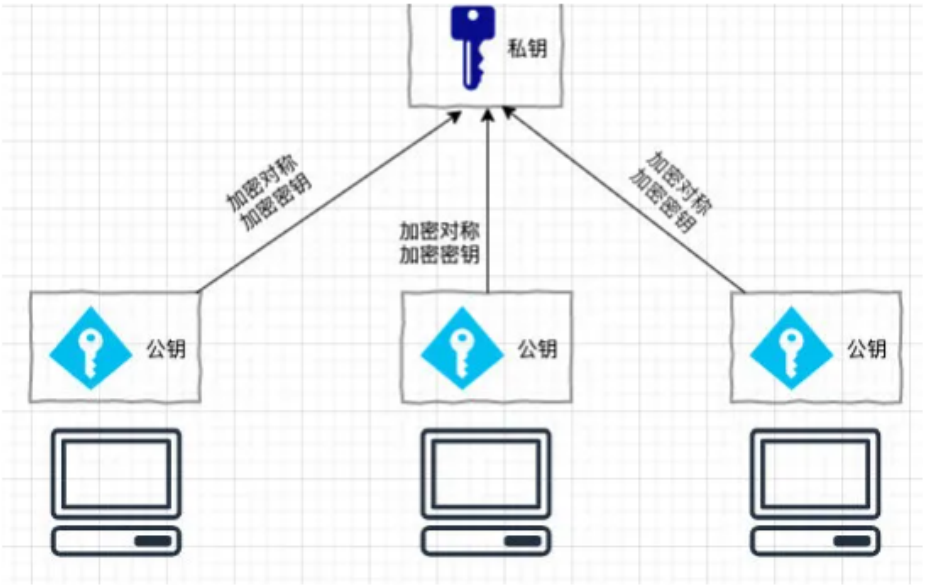
直接传输密钥无论从哪一端传从上节分析来看是不行了，这里我们再看另一种加密方式：**非对称加密**。

非对称加密即加解密双方使用不同的密钥，一把作为公钥，可以公开的，一把作为私钥，不能公开，公钥加密的密文只有私钥可以解密，私钥加密的内容，也只有公钥可以解密。

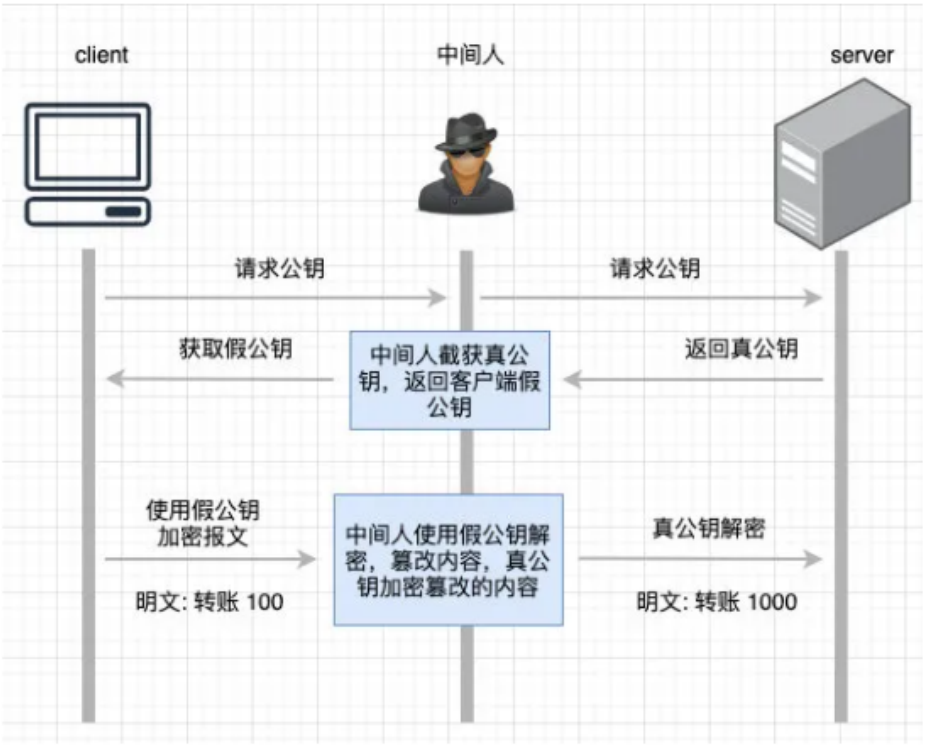
注：私钥加密其实这个说法其实并不严谨，准确的说私钥加密应该叫私钥签名，因为私密加密的信息公钥是可以解密的，而公钥是公开的，任何人都可以拿到，用公钥解密叫做验签

这样的话对于 server 来说，保管好私钥，发布公钥给其他 client，其他 client 只要把对称加密的密钥加密传给 server 即可，如此一来由于公钥加密只有私钥能解密，而私钥只有 server 有，所以能保证 client 向 server 传输是安全的，server 解密后即可拿到对称加密密钥，这样交换了密钥之后就可以用对称加密密钥通信了。



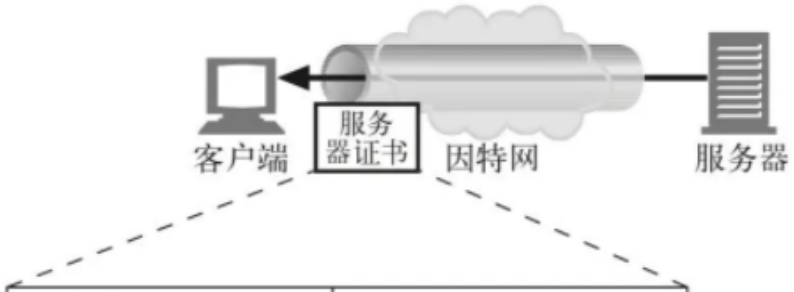


但是问题又来了， server 怎么把公钥安全地传输给 client 呢。如果直接传公钥，也会存在被中间人调包的风险。



数字证书，解决公钥传输信任问题

如何解决公钥传输问题呢，从现实生活中的场景找答案，员工入职时，企业一般会要求提供学历证明，显然不是什么阿猫阿狗的本本都可称为学历，这个学历必须由第三方权威机构（Certificate Authority，简称 CA）即教育部颁发，同理，server 也可以向 CA 申请证书，**在证书中附上公钥**，然后将证书传给 client，证书由站点管理者向 CA 申请，申请的时候会提交 DNS 主机名等信息，CA 会根据这些信息生成证书



证书序列号	35:DE:F4:CF
证书过期时间	Wed,Sep 17,2003
站点的组织名	Joe's Hardware Online
站点的DNS主机名	www.joes-hardware.com
站点的公开密钥	
证书颁发者的名称	RSA Data Security
证书颁发者的签名	John Doe

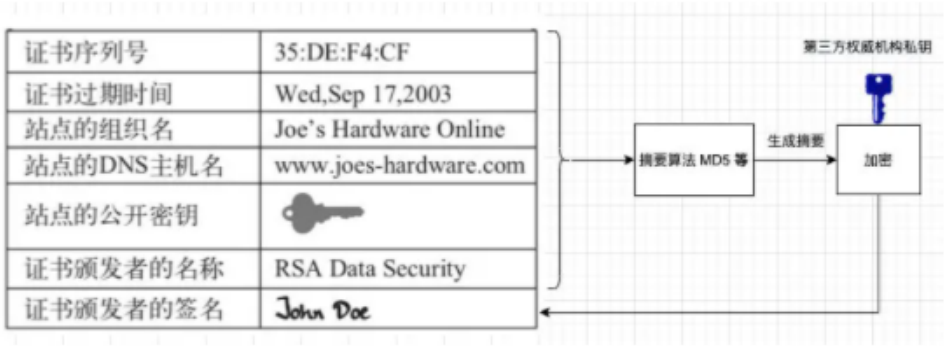
这样当 client 拿到证书后，就可以获得证书上的公钥，再用此公钥加密**对称加密密钥**传给 server 即可，看起来确实很完美，不过在这里大家要考虑两个问题

问题一、 如何验证证书的真实性，如何防止证书被篡改

想象一下上文中我们提到的学历，企业如何认定你提供的学历证书是真是假呢，答案是用

学历编号，企业拿到证书后用学历编号在学信网上一查就知道证书真伪了，学历编号其实就是我们常说的数字签名，可以防止证书造假。

回到 HTTPS 上，证书的数字签名该如何产生的呢，一图胜千言



步骤如下 1、 首先使用一些摘要算法（如 MD5）将证书明文（如证书序列号，DNS主机名等）生成摘要，然后再用第三方权威机构的私钥对生成的摘要进行加密（签名）

> 消息摘要是把任意长度的输入揉和而产生长度固定的伪随机输入的算法，无论输入的消息有多长，计算出来的消息摘要的长度总是固定的，一般来说，只要内容不同，产生的摘要必然不同（相同的概率可以认为接近于 0），所以可以验证内容是否被篡改了。

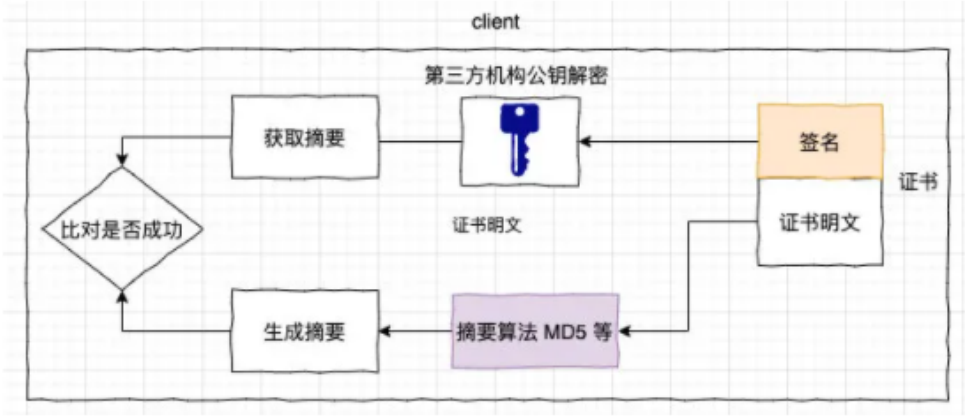
为啥要先生成摘要再加密呢，不能直接加密？

因为使用非对称加密是非常耗时的，如果把整个证书内容都加密生成签名的话，客户端验验签也需要把签名解密，证书明文较长，客户端验签就需要很长的时间，而用摘要的话，会把内容很长的明文压缩成小得多的定长字符串，客户端验签的话就会快得多。

2、客户端拿到证书后也用同样的摘要算法对证书明文计算摘要，两者一笔对就可以发现报文是否被篡改了，那为啥要用第三方权威机构（Certificate Authority，简称 CA）私钥对摘要加密呢，因为摘要算法是公开的，中间人可以替换掉证书明文，再根据证书上的摘要算法计算出摘要后把证书上的摘要也给替换掉！这样 client 拿到证书后计算摘要发现一样，

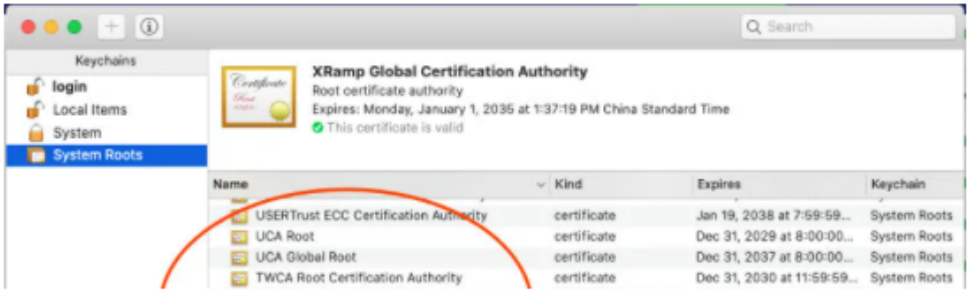
误以为此证书是合法就中招了。所以必须要用 CA 的私钥给摘要进行加密生成签名，这样的话 client 得用 CA 的公钥来给签名解密，拿到的才是未经篡改合法的摘要（私钥签名，公钥才能解密）

server 将证书传给 client 后，client 的验签过程如下



这样的话，由于只有 CA 的公钥才能解密签名，如果客户端收到一个假的证书，使用 CA 的公钥是无法解密的，如果客户端收到了真的证书，但证书上的内容被篡改了，摘要比对不成功的话，客户端也会认定此证书非法。

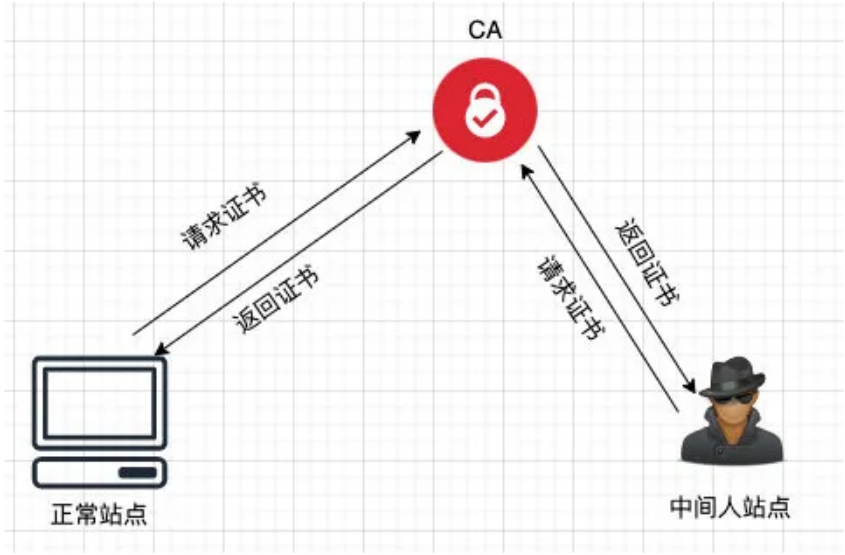
细心的你一定发现了问题，CA 公钥如何安全地传输到 client？如果还是从 server 传输到 client，依然无法解决公钥被调包的风险，**实际上此公钥是存在于 CA 证书上，而此证书（也称 Root CA 证书）被操作系统信任，内置在操作系统上的，无需传输**，如果用的是 Mac 的同学，可以打开 keychain 查看一下，可以看到很多内置的被信任的证书。



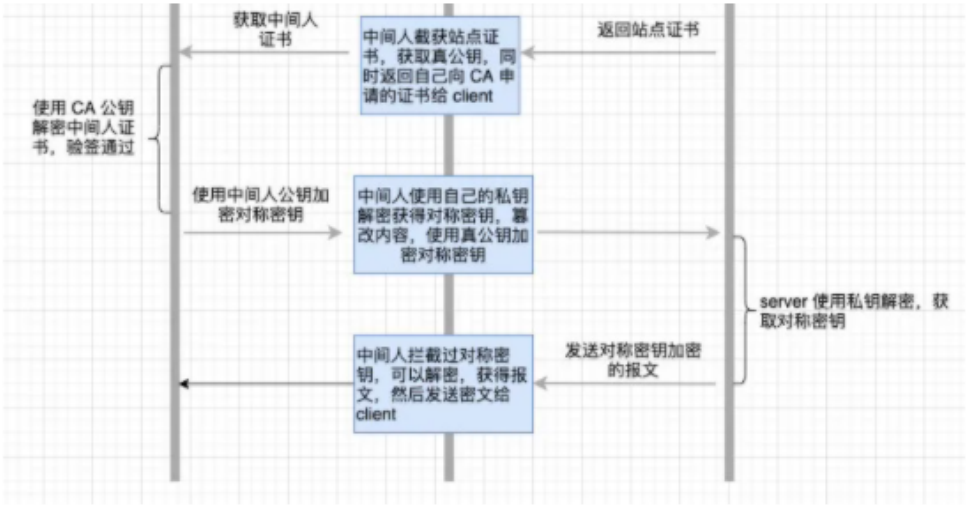
server 传输 CA 颁发的证书，客户中收到证书后使用**内置 CA 证书中的公钥**来解密签名，验签即可，这样的话就解决了公钥传输过程中被调包的风险。

问题二、 如何防止证书被调包

实际上任何站点都可以向第三方权威机构申请证书，中间人也不例外。

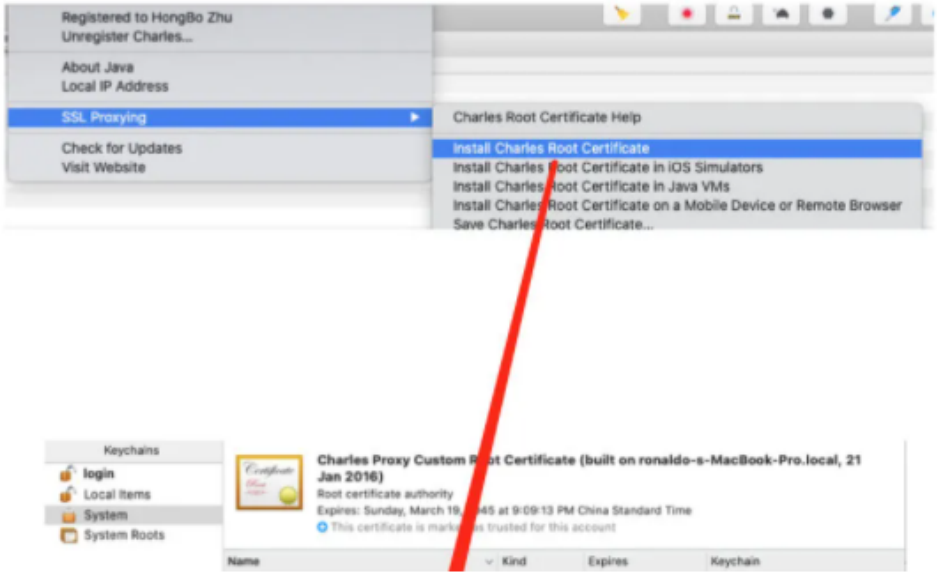


正常站点和中间人都可以向 CA 申请证书，获得认证的证书由于都是 CA 颁发的，所以都是合法的，那么此时中间人是否可以在传输过程中将正常站点发给 client 的证书替换成自己的证书呢，如下所示

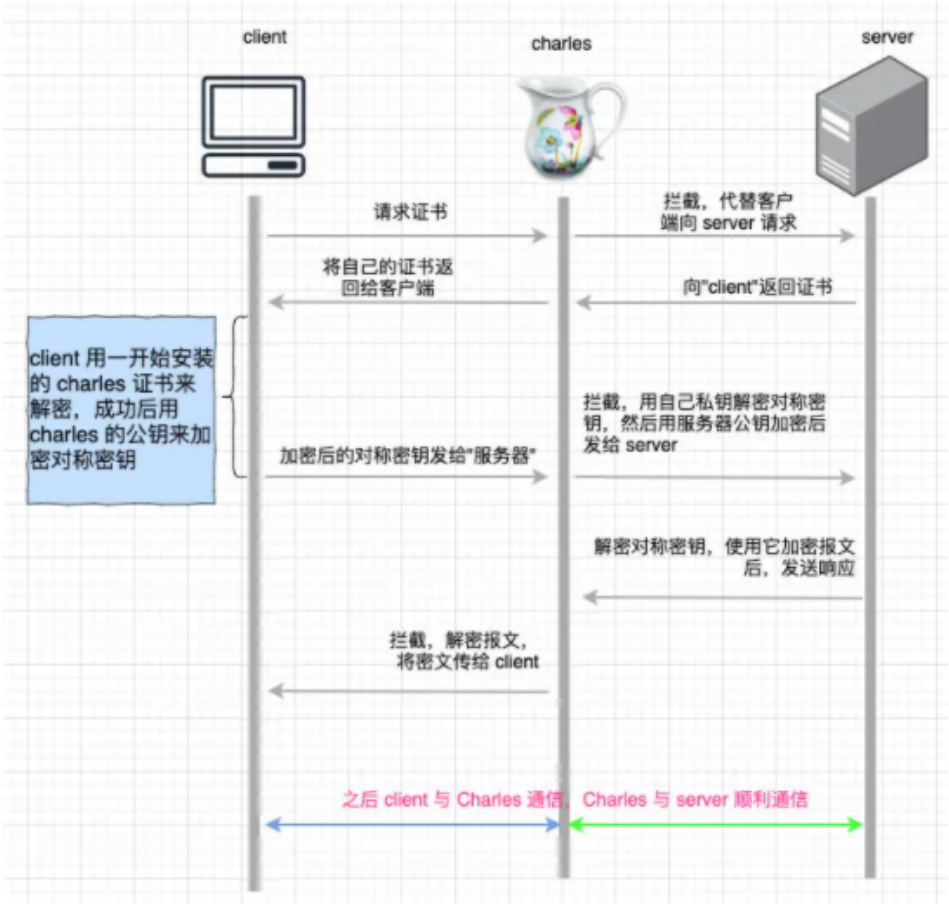


答案是不行，因为客户端除了通过验签的方式验证证书是否合法之外，**还需要验证证书上的域名与自己的请求域名是否一致**，中间人中途虽然可以替换自己向 CA 申请的合法证书，但此证书中的域名与 client 请求的域名不一致，client 会认定为不通过！

但是上面的证书调包给了我们一种思路，什么思路？大家想想， HTTPS 既然是加密的，charles 这些「中间人」为啥能抓到明文的包呢，其实就是用了证书调包这一手法，想想看，在用 charles 抓 HTTPS 的包之前我们先要做什么，当然是安装 charles 的证书



这个证书里有 charles 的公钥，这样的话 charles 就可以将 server 传给 client 的证书调包成自己的证书，client 拿到后就可以用你安装的 charles 证书来验签等，验证通过之后就会用 charles 证书中的公钥来加密对称密钥了，整个流程如下



由此可知，charles 这些中间人能抓取 HTTPS 包的前提是信任它们的 CA 证书，然后就可以通过替换证书的方式进行瞒天过海，所以我们千万不要随便信任第三方的证书，避免安全风险。

其它 HTTPS 相关问题

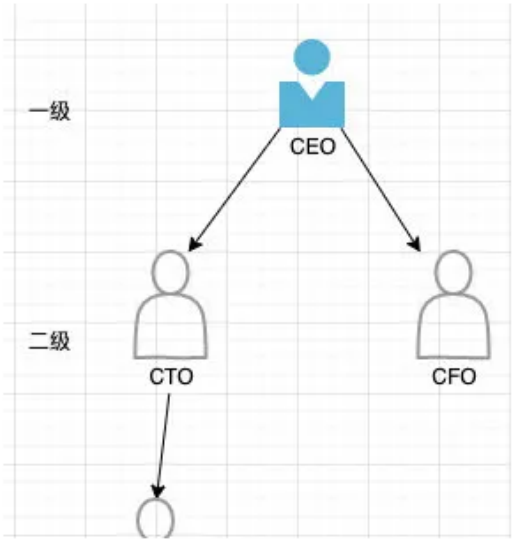
> 什么是双向认证

以上的讲述过程中，我们只是在 client 端验证了 server 传输证书的合法性，但 server 如何验证 client 的合法性，还是用证书，我们在网上进行转账等操作时，想想看是不是要先将银行发给我们的 U 盾插到电脑上？其实也是因为 U 盾内置了证书，通信时将证书发给 server，server 验证通过之后即可开始通信。

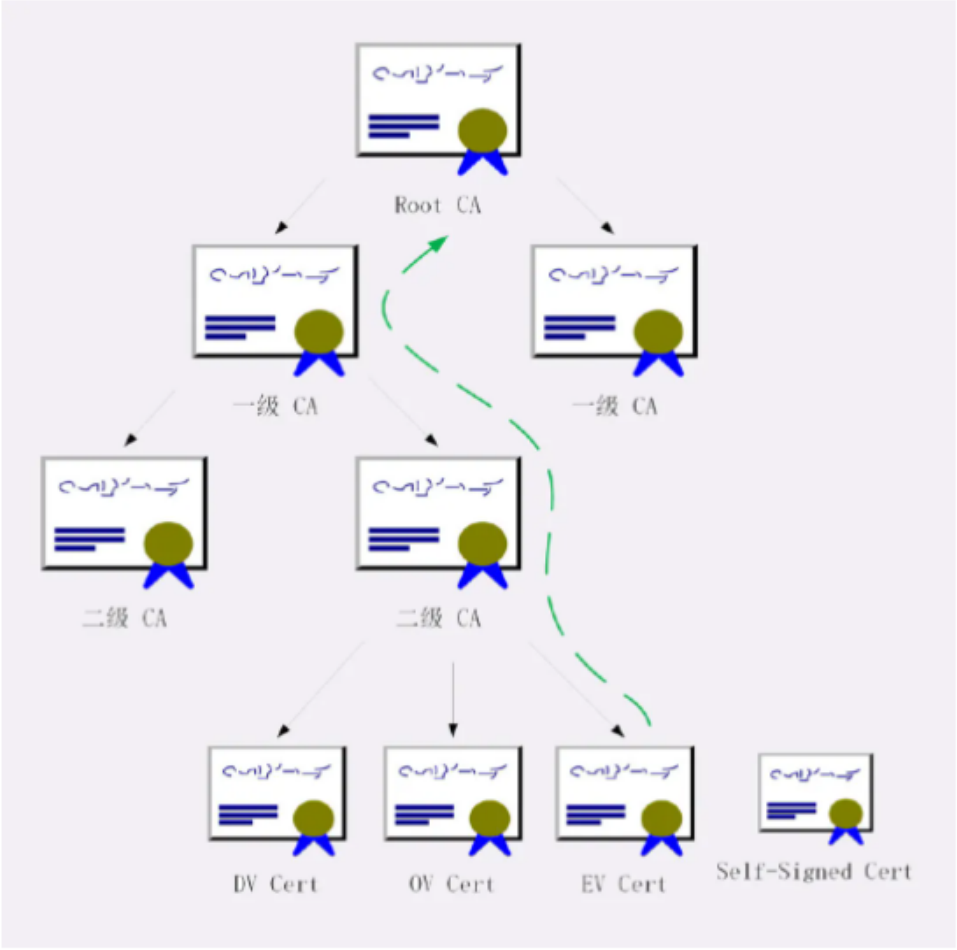
画外音：身份认证只是 U 盾功能的一种，还有其他功能，比如加解密都是在 U 盾中执行，保证了密钥不会出现在内存中

> 什么是证书信任链

前文说了，我们可以向 CA 申请证书，但全世界的顶级 CA（Root CA） 就那么几个，每天都有很多人要向它申请证书，它也忙不过来啊，怎么办呢，想想看在一个公司里如果大家都找 CEO 办事，他是不是要疯了，那他能怎么办？授权，他会把权力交给 CTO，CFO 等，这样你们只要把 CTO 之类的就行了，CTO 如果也忙不过来呢，继续往下授权啊。



同样的，既然顶级 CA 忙不过来，那它就向下一级，下下级 CA 授权即可，这样我们就只要找一级/二级/三级 CA 申请证书即可。怎么证明这些证书被 Root CA 授权过了呢，小一点的 CA 可以让大一点的 CA 来签名认证，比如一级 CA 让 Root CA 来签名认证，二级 CA 让一级 CA 来签名认证,Root CA 没有人给他签名认证，只能自己证明自己了，这个证书就叫「自签名证书」或者「根证书」，我们必须信任它，不然证书信任链是走不下去的（这个根证书前文我们提过，其实是内置在操作系统中的）



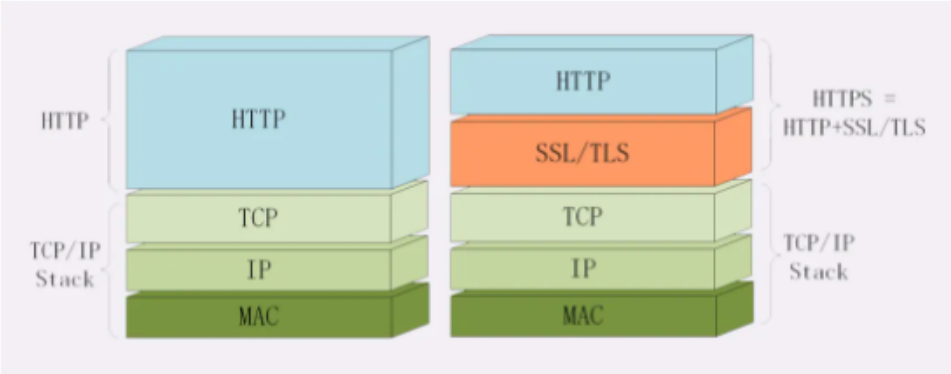
证书信任链

现在我们看看如果站点申请的是二级 CA 颁发的证书，client 收到之后会如何验证这个证书呢，实际上 service 传了传给二级 CA 的证书外，**还会把证书信任链也一起传给客户端**，这样客户端会按如下步骤进行验证：

1. 浏览器就使用信任的根证书（根公钥）解析证书链的根证书得到一级证书的公钥+摘要验签
2. 拿一级证书的公钥解密一级证书，拿到二级证书的公钥和摘要验签
3. 再然后拿二级证书的公钥解密 server 传过来的二级证书，得到服务器的公钥和摘要验签，验证过程就结束了

总结

相信大家看完本文应该对 HTTPS 的原理有了很清楚的认识了， HTTPS 无非就是 HTTP + SSL/TLS



而 SSL/TLS 的功能其实本质上是**如何协商出安全的对称加密密钥以利用此密钥进行后续通讯的过程**，带着这个疑问相信你不难理解数字证书和数字签名这两个让人费解的含义，搞懂了这些也就明白了为啥 HTTPS 是加密的，charles 这些工具却能抓包出明文来。

巨人的肩膀

- https://juejin.cn/post/6844903958863937550
- https://showme.codes/2017-02-20/understand-https/
- 极客时间，透视 HTTP 协议

- https://zhuanlan.zhihu.com/p/67199487

推荐阅读 点击标题可跳转

HTTPS 的工作原理

看完这篇 HTTPS，和面试官扯皮就没问题了

HTTPS 原理分析——带着疑问层层深入