也许，这样理解HTTPS更容易

程序君 程序员大咖 Yesterday

Linux编程

点击右侧关注，免费入门到精通！

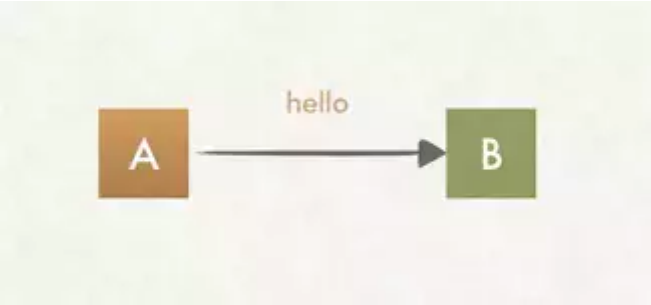
作者丨会编程的娃娃鱼

https://juejin.im/post/5b8367b1e51d453884361fc3

⛵摘要：

本文尝试一步步还原HTTPS的设计过程，以理解为什么HTTPS最终会是这副模样。但是这并不代表HTTPS的真实设计过程。在阅读本文时，你可以尝试放下已有的对HTTPS的理解，这样更利于“还原”过程。

我们先不聊HTTP，HTTPS，我们先从一个聊天软件说起，我们要实现A能发一个hello消息给B：



如果我们要实现这个聊天软件，本文只考虑安全性问题，要实现

> A发给B的hello消息包，即使被中间人拦截到了，也无法得知消息的内容

⛵如何做到真正的安全?

这个问题，很多人马上就想到了各种加密算法，什么对称加密、非对称加密、DES、RSA、XX、噼里啪啦~

而我想说，加密算法只是解决方案，我们首先要做的是理解我们的问题域——什么是安全?我个人的理解是:

> A与B通信的内容，有且只有A和B有能力看到通信的真正内容

好，问题域已经定义好了（现实中当然不止这一种定义）。对于解决方案，很容易就想到了对消息进行加密。

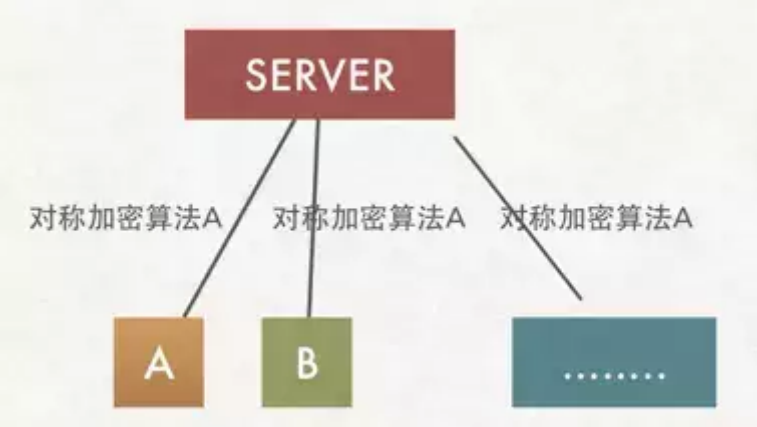
题外话，但是只有这一种方法吗？我看未必，说不定在将来会出现一种物质打破当前世界的通信假设，实现真正意义上的保密。

对于A与B这样的简单通信模型，我们很容易做出选择:

这就是对称加密算法，其中图中的密钥S同时扮演加密和解密的角色。具体细节不是本文范畴。

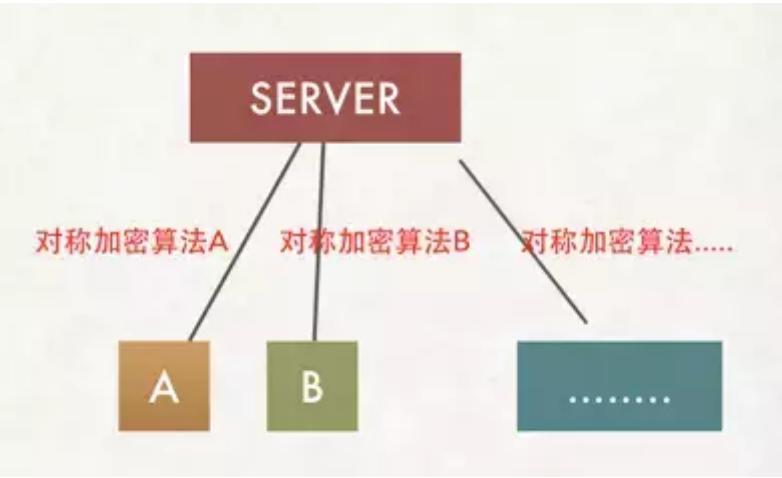
只要这个密钥S不公开给第三者，同时密钥S足够安全，我们就解决了我们一开始所定问题域了。因为世界上有且只有A与B知道如何加密和解密他们之间的消息。

但是，在WWW环境下，我们的Web服务器的通信模型没有这么简单：



如果服务器端对所有的客户端通信都使用同样的对称加密算法，无异于没有加密。那怎么办呢？即能使用对称加密算法，又不公开密钥？请读者思考21秒钟。😜

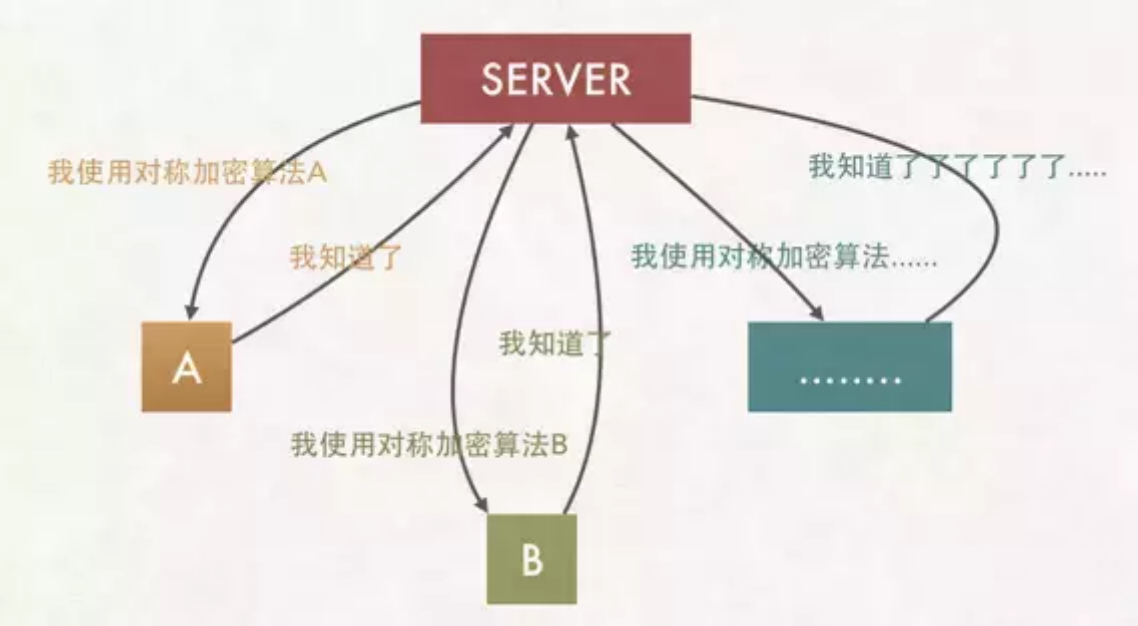
答案是：Web服务器与每个客户端使用不同的对称加密算法：



⛵如何确定对称加密算法

慢着，另一个问题来了，我们的服务器端怎么告诉客户端该使用哪种对称加密算法？

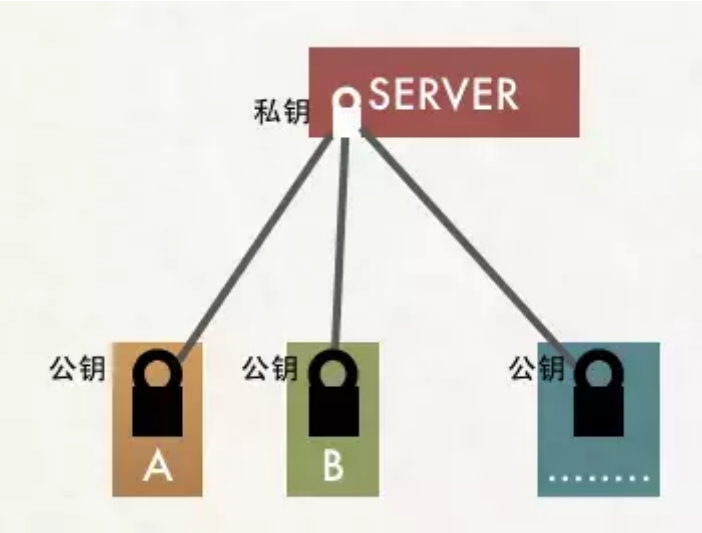
当然是通过协商。



但是，你协商的过程是没有加密的，还是会被中间人拦截。那我们再对这个协商过程进行对称加密就好了，那你对协商过程加密的加密还是没有加密，怎么办？再加密不就好了……好吧，进行鸡生蛋蛋生鸡的问题了。

⛵如何对协商过程进行加密

新问题来了，如何对协商过程进行加密？密码学领域中，有一种称为“非对称加密”的加密算法，特点是私钥加密后的密文，只要是公钥，都可以解密，但是公钥加密后的密文，只有私钥可以解密。私钥只有一个人有，而公钥可以发给所有的人。



虽然服务器端向A、B……的方向还是不安全的，但是至少A、B向服务器端方向是安全的。

好了，如何协商加密算法的问题，我们解决了：使用非对称加密算法进行对称加密算法协商过程。

这下，你明白为什么HTTPS同时需要对称加密算法和非对称加密算法了吧?

协商什么加密算法

要达到Web服务器针对每个客户端使用不同的对称加密算法，同时，我们也不能让第三者知道这个对称加密算法是什么，怎么办?

使用随机数，就是使用随机数来生成对称加密算法。这样就可以做到服务器和客户端每次交互都是新的加密算法、只有在交互的那一该才确定加密算法。

这下，你明白为什么HTTPS协议握手阶段会有这么多的随机数了吧。

如何得到公钥?

细心的人可能已经注意到了如果使用非对称加密算法，我们的客户端A，B需要一开始就持有公钥，要不没法开展加密行为啊。

这下，我们又遇到新问题了，如何让A、B客户端安全地得到公钥?

我能想到的方案只有这些：

方案1. 服务器端将公钥发送给每一个客户端

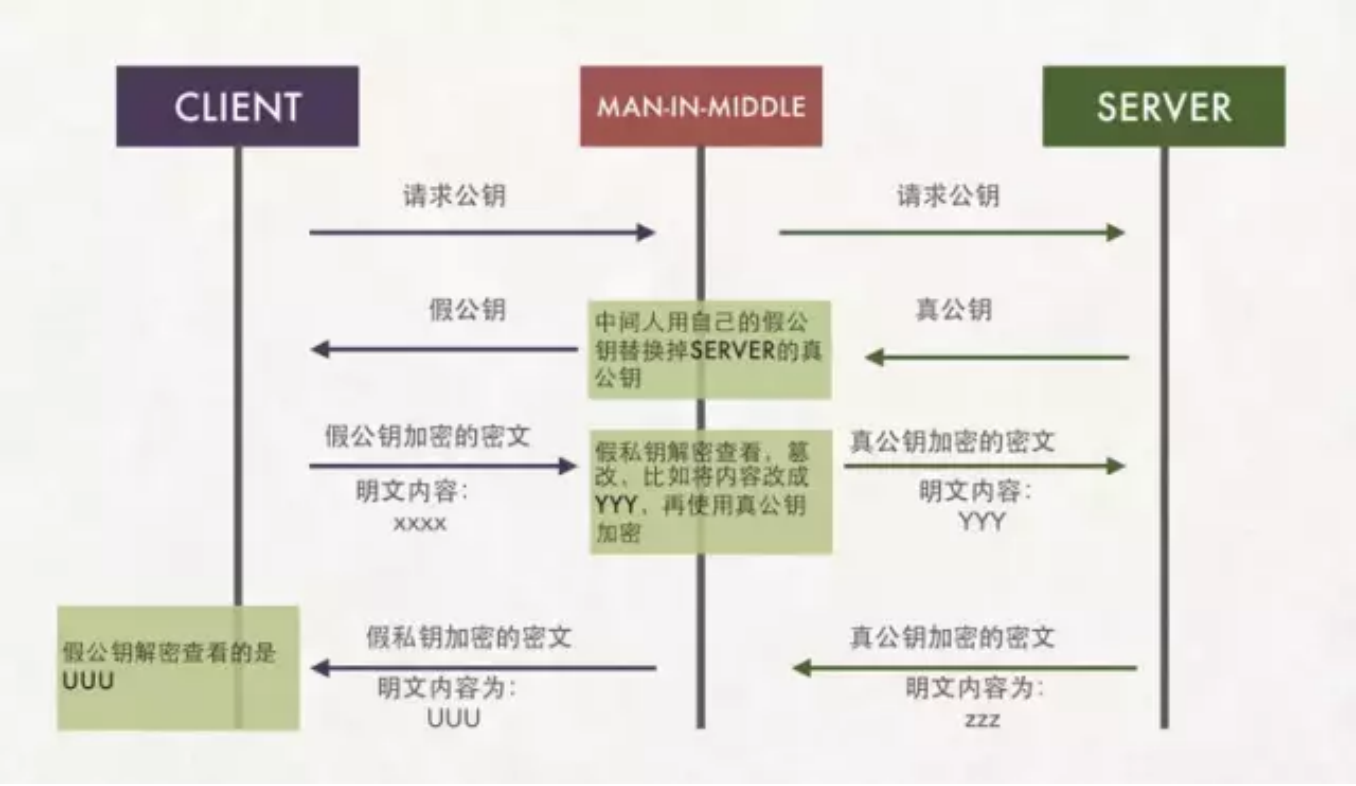
方案2. 服务器端将公钥放到一个远程服务器，客户端可以请求得到

我们选择方案1，因为方案2又多了一次请求，还要另外处理公钥的放置问题。

公钥被调包了怎么办？又是一个鸡生蛋蛋生鸡问题？

但是方案1有个问题：如果服务器端发送公钥给客户端时，被中间人调包了，怎么办？

我画了张图方便理解：



显然，让每个客户端的每个浏览器默认保存所有网站的公钥是不现实的。

⛵使用第三方机构的公钥解决鸡生蛋蛋生鸡问题

公钥被调包的问题出现，是因为我们的客户端无法分辨返回公钥的人到底是中间人，还是真的服务器。这其实就是密码学中提的身份验证问题。

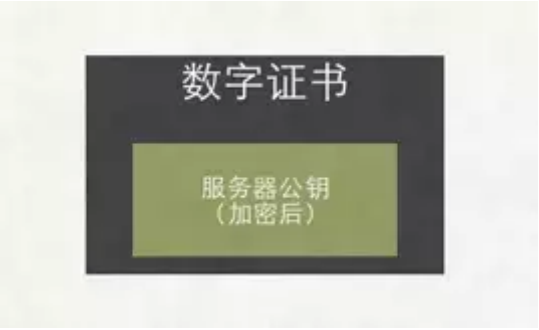
如果让你来解决，你怎么解决？如果你了解过HTTPS，会知道使用数字证书来解决。但是你想过证书的本质是什么？请放下你对HTTPS已有的知识，自己尝试找到解决方案。

我是这样解决的。既然服务器需要将公钥传给客户端，这个过程本身是不安全，那么我们为什么不对这个过程本身再加密一次？可是，你是使用对称加密，还是非对称加密？这下好了，我感觉又进了鸡生蛋蛋生鸡问题了。

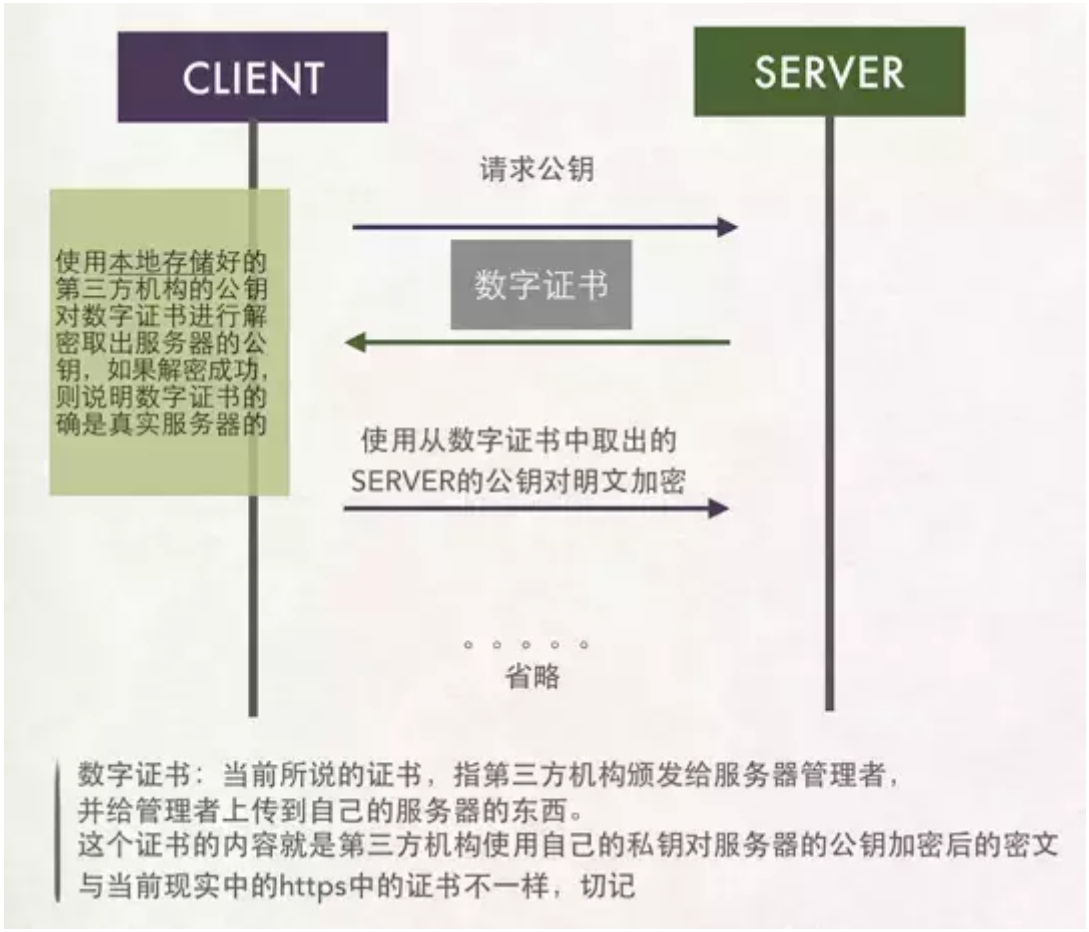
问题的难点是如果我们选择直接将公钥传递给客户端的方案，我们始终无法解决公钥传递被中间人调包的问题。

所以，我们不能直接将服务器的公钥传递给客户端，而是第三方机构使用它的私钥对我们的公钥进行加密后，再传给客户端。客户端再使用第三方机构的公钥进行解密。

下图就是我们设计的第一版“数字证书”，证书中只有服务器交给第三方机构的公钥，而且这个公钥被第三方机构的私钥加密了：



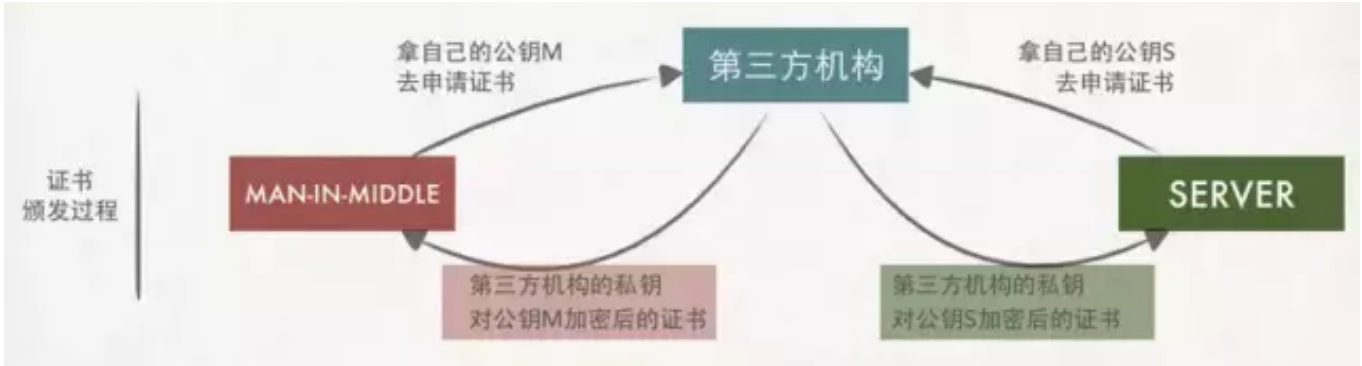
如果能解密，就说明这个公钥没有被中间人调包。因为如果中间人使用自己的私钥加密后的东西传给客户端，客户端是无法使用第三方的公钥进行解密的。



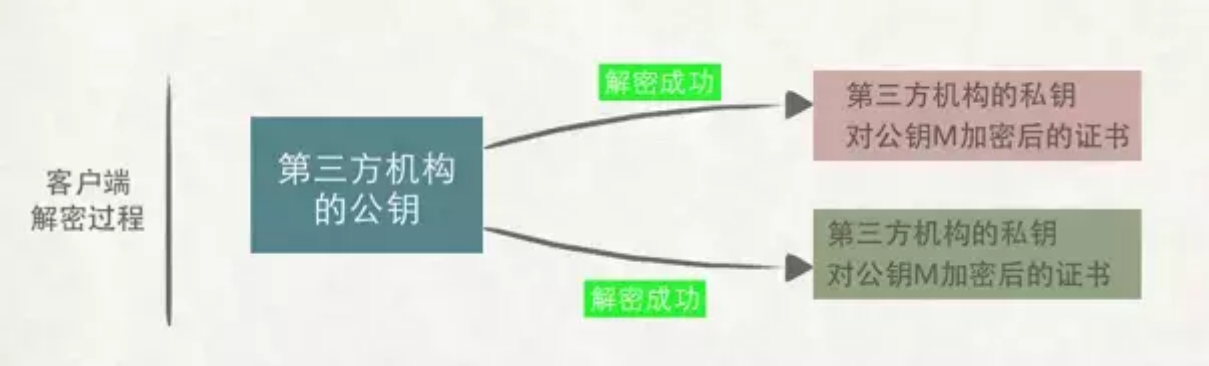
话到此，我以为解决问题了。但是现实中HTTPS，还有一个数字签名的概念，我没法理解它的设计理由。

原来，我漏掉了一个场景：第三方机构不可能只给你一家公司制作证书，它也可能会给中间人这样有坏心思的公司发放证书。这样的，中间人就有机会对你的证书进行调包，客户端在这种情况下是无法分辨出是接收的是你的证书，还是中间人的。因为不论中间人，还是你的证书，都能使用第三方机构的公钥进行解密。像下面这样：

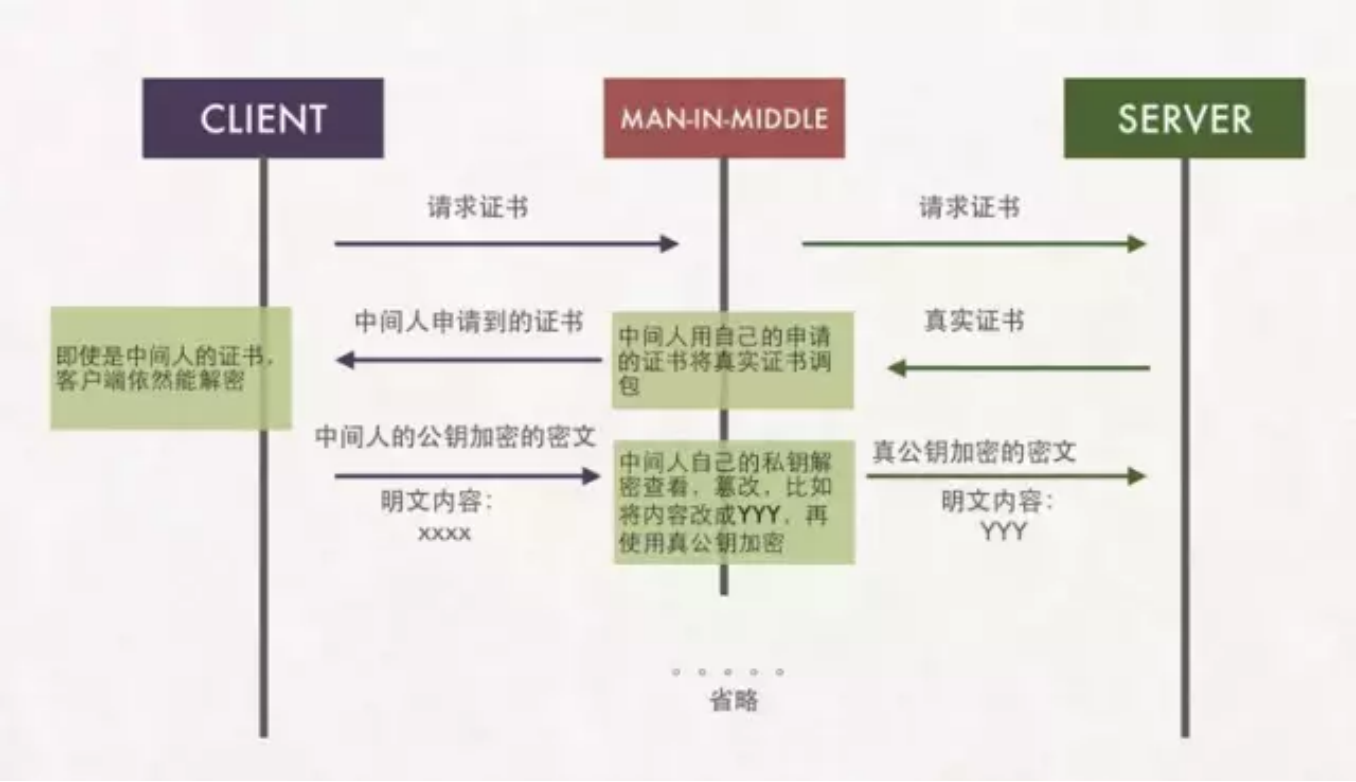
第三方机构向多家公司颁发证书的情况：



客户端能解密同一家第三机构颁发的所有证书：



最终导致其它持有同一家第三方机构证书的中间人可以进行调包：



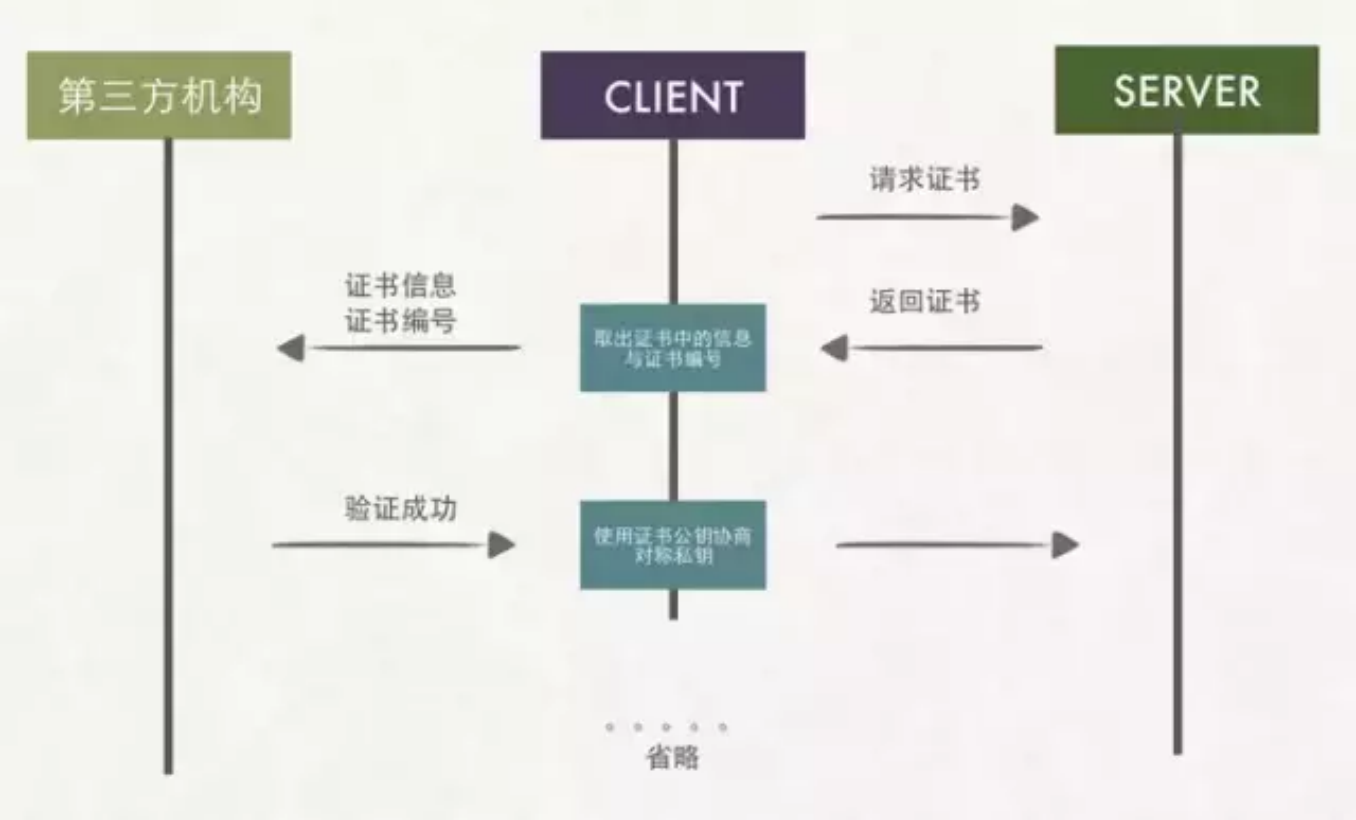
数字签名，解决同一机构颁发的不同证书被篡改问题

要解决这个问题，我们首先要想清楚一个问题，辨别同一机构下不同证书的这个职责，我们应该放在哪？

只能放到客户端了。意思是，客户端在拿到证书后，自己就有能力分辨证书是否被篡改了。如何才能有这个能力呢？

我们从现实中找灵感。比如你是HR，你手上拿到候选人的学历证书，证书上写了持证人，颁发机构，颁发时间等等，同时证书上，还写有一个最重要的：证书编号！我们怎么鉴别这张证书是的真伪呢？只要拿着这个证书编号上相关机构去查，如果证书上的持证人与现实的这个候选人一致，同时证书编号也能对应上，那么就说明这个证书是真实的。

我们的客户端能不能采用这个机制呢？像这样：



可是，这个“第三方机构”到底是在哪呢？是一个远端服务？不可能吧？如果是个远端服务，整个交互都会慢了。所以，这个第三方机构的验证功能只能放在客户端的本地了。

⛵客户端本地怎么验证证书呢？

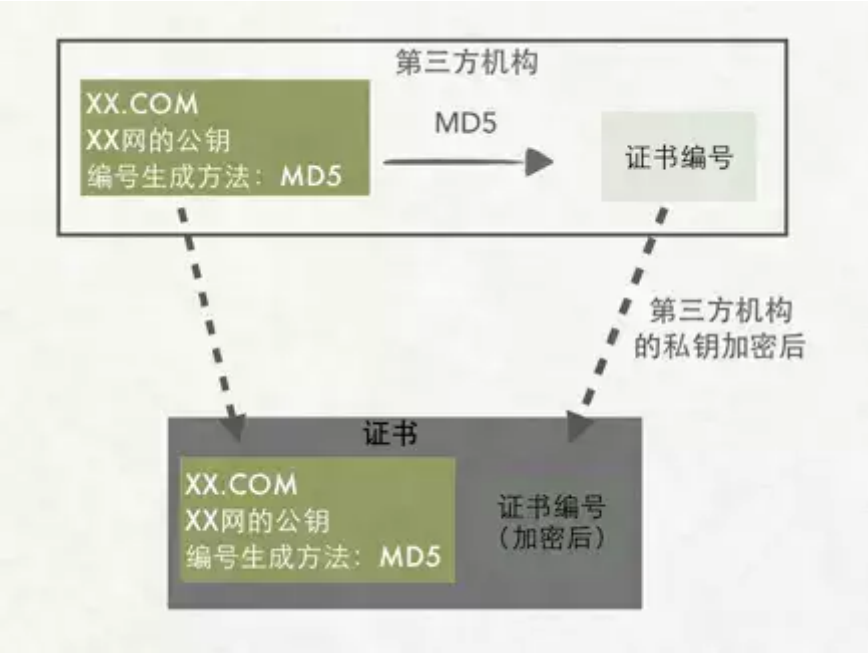
客户端本地怎么验证证书呢？答案是证书本身就已经告诉客户端怎么验证证书的真伪。

也就是证书上写着如何根据证书的内容生成证书编号。客户端拿到证书后根据证书上的方法自己生成一个证书编号，如果生成的证书编号与证书上的证书编号相同，那么说明这个证书是真实的。

同时，为避免证书编号本身又被调包，所以使用第三方的私钥进行加密。

这地方有些抽象，我们来个图帮助理解：

证书的制作如图所示。证书中的“编号生成方法MD5”就是告诉客户端：你使用MD5对证书的内容求值就可以得到一个证书编号。



当客户端拿到证书后，开始对证书中的内容进行验证，如果客户端计算出来的证书编号与证书中的证书编号相同，则验证通过：

但是第三方机构的公钥怎么跑到了客户端的机器中呢？世界上这么多机器。

其实呢，现实中，浏览器和操作系统都会维护一个权威的第三方机构列表（包括它们的公钥）。因为客户端接收到的证书中会写有颁发机构，客户端就根据这个颁发机构的值在本地

找相应的公钥。

> 题外话：如果浏览器和操作系统这道防线被破了，就没办法。想想当年自己装过的非常规XP系统，都害怕。

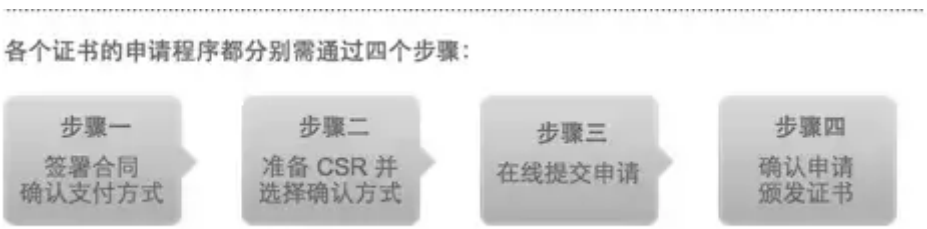
说到这里，想必大家已经知道上文所说的，证书就是HTTPS中数字证书，证书编号就是数字签名，而第三方机构就是指数字证书签发机构（CA）。

⛵CA如何颁发数字证书给服务器端的?

当我听到这个问题时，我误以为，我们的SERVER需要发网络请求到CA部门的服务器来拿这个证书。😭 到底是我理解能力问题，还是。。

其实，问题应该是CA如何颁发给我们的网站管理员，而我们的管理员又如何将这个数字证书放到我们的服务器上。

我们如何向CA申请呢？每个CA机构都大同小异，我在网上找了一个：



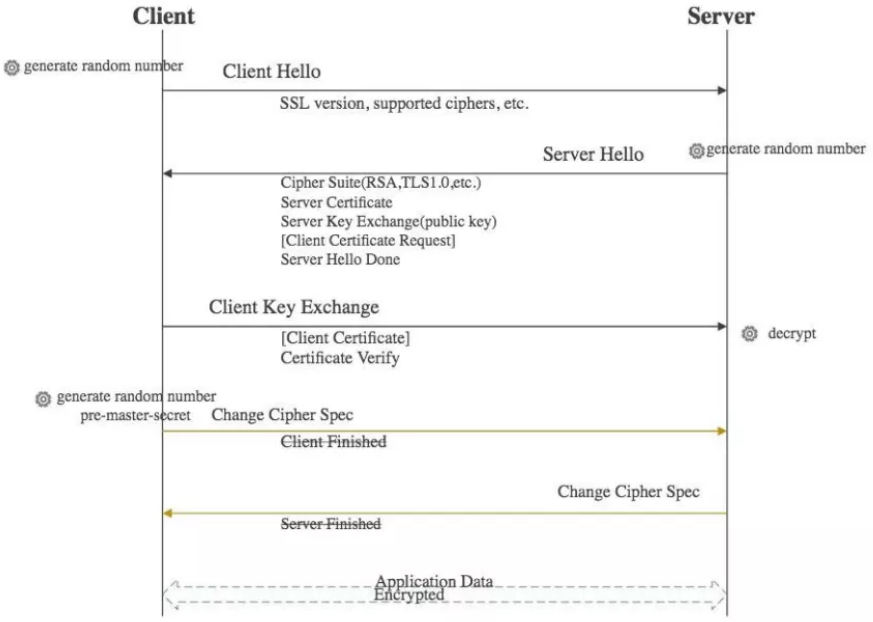
拿到证书后，我们就可以将证书配置到自己的服务器上了。那么如何配置？这是具体细节了，留给大家google了。

⛵也许我们需要整理一下思路

我们通过推算的方式尝试还原HTTPS的设计过程。这样，我们也就明白了为什么HTTPS比HTTP多那么多次的交互，为什么HTTPS的性能会差，以及找到HTTPS的性能优化点。

而上面一大堆工作都是为了让客户端与服务器端安全地协商出一个对称加密算法。这就是HTTPS中的SSL/TLS协议主要干的活。剩下的就是通信时双方使用这个对称加密算法进行加密解密。

以下是一张HTTPS协议的真实交互图（从网上copy的，忘了从哪了，如果侵权麻烦告知）：



⛵能不能用一句话总结HTTPS?

答案是不能，因为HTTPS本身实在太复杂。但是我还是尝试使用一段话来总结HTTPS:

HTTPS要使客户端与服务器端的通信过程得到安全保证，必须使用的对称加密算法，但是协商对称加密算法的过程，需要使用非对称加密算法来保证安全，然而直接使用非对称加密的过程本身也不安全，会有中间人篡改公钥的可能性，所以客户端与服务器不直接使用公钥，而是使用数字证书签发机构颁发的证书来保证非对称加密过程本身的安全。这样通过这些机制协商出一个对称加密算法，就此双方使用该算法进行加密解密。从而解决了客户端与服务器端之间的通信安全问题。

好长的一段话。

⛵后记

以上是个人为理解HTTPS而编造出来的自圆其说的看法。顶多只能算是HTTPS的科普文章。如有错误，请指出，万分感谢。

那么，我为什么会觉得以这种方式理解HTTPS会更容易呢？我个人给出的答案是：当你自己为一家人做一次菜时，你就会理解妈妈天天做菜的不易了。

推荐↓↓↓

长 按 关 注

👉【16个技术公众号】都在这里！

涵盖：程序员大咖、源码共读、程序员共读、数据结构与算法、黑客技术和网络安全、大数据科技、编程前端、Java、Python、Web编程开发、Android、iOS开发、Linux、数据库研发、幽默程序员等。

Read more