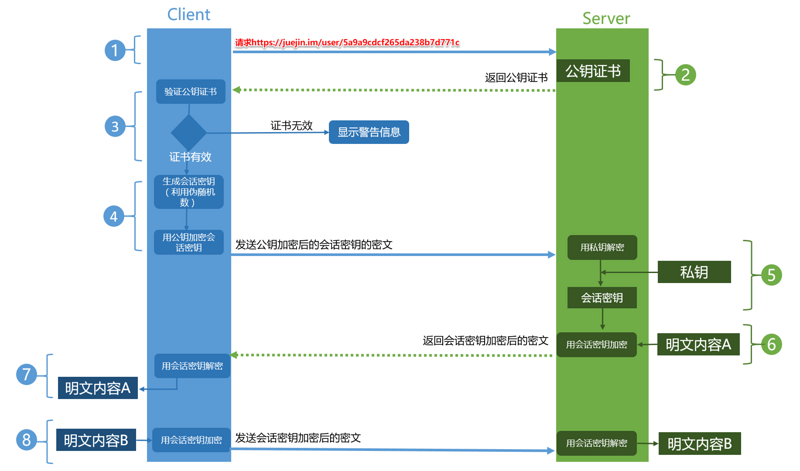
**HTTPS工作流程**



1.Client发起一个HTTPS（比如https://juejin.im/user/5a9a9cdcf265da238b7d771c）的请求，根据RFC2818的规定，Client知道需要连接Server的443（默认）端口。

2.Server把事先配置好的公钥证书（public key certificate）返回给客户端。

3.Client验证公钥证书：比如是否在有效期内，证书的用途是不是匹配Client请求的站点，是不是在CRL吊销列表里面，它的上一级证书是否有效，这是一个递归的过程，直到验证到根证书（操作系统内置的Root证书或者Client内置的Root证书）。如果验证通过则继续，不通过则显示警告信息。

4.Client使用伪随机数生成器生成加密所使用的对称密钥，然后用证书的公钥加密这个对称密钥，发给Server。

5.Server使用自己的私钥（private key）解密这个消息，得到对称密钥。至此，Client和Server双方都持有了相同的对称密钥。

6.Server使用对称密钥加密“明文内容A”，发送给Client。

7.Client使用对称密钥解密响应的密文，得到“明文内容A”。

8.Client再次发起HTTPS的请求，使用对称密钥加密请求的“明文内容B”，然后Server使用对称密钥解密密文，得到“明文内容B”。

**一.解决内容可能被窃听的问题——加密**

**方法 ，对称加密+非对称加密(HTTPS采用这种方式)**

使用对称密钥的好处是解密的效率比较快，使用非对称密钥的好处是可以使得传输的内容不能被破解，因为就算你拦截到了数据，但是没有对应的私钥，也是不能破解内容的。就比如说你抢到了一个保险柜，但是没有保险柜的钥匙也不能打开保险柜。那我们就将对称加密与非对称加密结合起来,充分利用两者各自的优势，**在交换密钥环节使用非对称加密方式，之后的建立通信交换报文阶段则使用对称加密方式**。

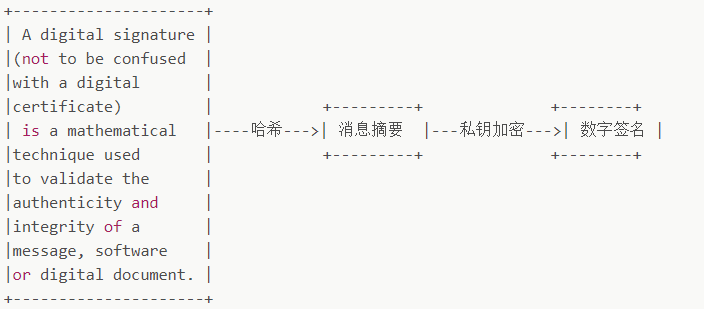
具体做法是：**发送密文的一方使用对方的公钥进行加密处理“对称的密钥”，然后对方用自己的私钥解密拿到“对称的密钥”，这样可以确保交换的密钥是安全的前提下，使用对称加密方式进行通信**。所以，HTTPS采用对称加密和非对称加密两者并用的混合加密机制。

**二.解决报文可能遭篡改问题——数字签名**

网络传输过程中需要经过很多中间节点，虽然数据无法被解密，但可能被篡改，那如何校验数据的完整性呢？----校验数字签名。

**数字签名有两种功效**：

* 能确定消息确实是由发送方签名并发出来的，因为别人假冒不了发送方的签名。
* 数字签名能确定消息的完整性,证明数据是否未被篡改过。

**数字签名如何生成:**  


将一段文本先用Hash函数生成消息摘要，然后用发送者的私钥加密生成数字签名，与原文文一起传送给接收者。接下来就是接收者校验数字签名的流程了。

**校验数字签名流程**：

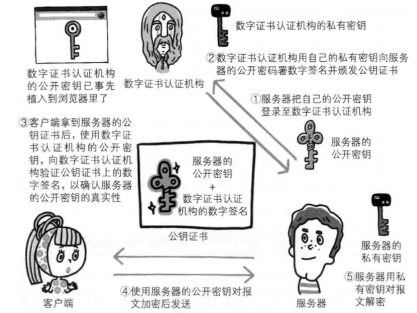


接收者只有用发送者的公钥才能解密被加密的摘要信息，然后用HASH函数对收到的原文产生一个摘要信息，与上一步得到的摘要信息对比。如果相同，则说明收到的信息是完整的，在传输过程中没有被修改，否则说明信息被修改过，因此数字签名能够验证信息的完整性。

假设消息传递在Kobe，James两人之间发生。James将消息连同数字签名一起发送给Kobe，Kobe接收到消息后，通过校验数字签名，就可以验证接收到的消息就是James发送的。当然，这个过程的前提是Kobe知道James的公钥。问题的关键的是，和消息本身一样，公钥不能在不安全的网络中直接发送给Kobe,或者说拿到的公钥如何证明是James的。

此时就需要引入了**证书颁发机构**（Certificate Authority，简称CA），CA数量并不多，**Kobe客户端内置了所有受信任CA的证书**。CA对James的公钥（和其他信息）数字签名后生成证书。

**三.解决通信方身份可能被伪装的问题——数字证书**

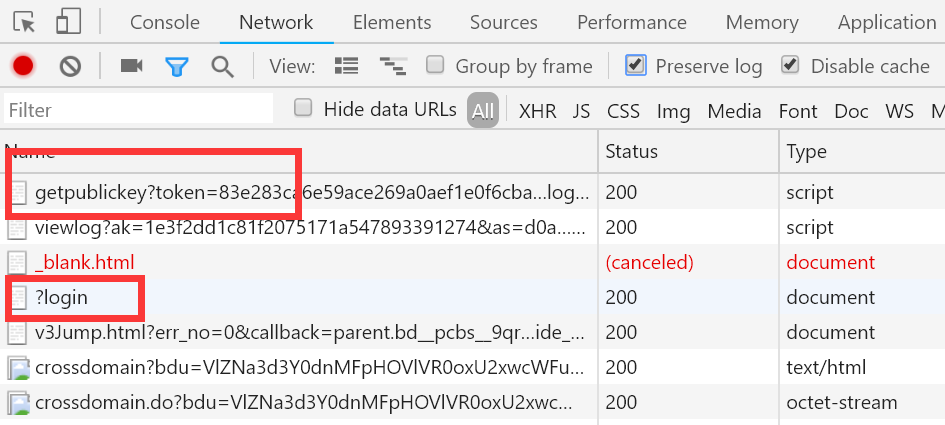
数字证书认证机构处于客户端与服务器双方都可信赖的第三方机构的立场上。  
  
我们来介绍一下数字证书认证机构的业务流程：

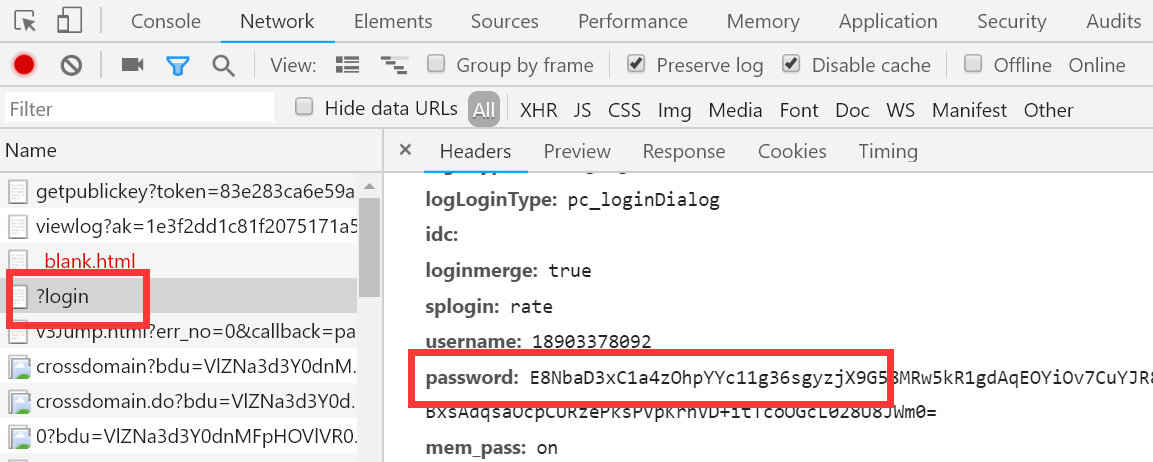
* 服务器的运营人员向第三方机构CA提交公钥、组织信息、个人信息(域名)等信息并申请认证;
* CA通过线上、线下等多种手段验证申请者提供信息的真实性，如组织是否存在、企业是否合法，是否拥有域名的所有权等;
* 如信息审核通过，CA会向申请者签发认证文件-证书。证书包含以下信息：申请者公钥、申请者的组织信息和个人信息、签发机构 CA的信息、有效时间、证书序列号等信息的明文，同时包含一个签名。 其中签名的产生算法：首先，使用散列函数计算公开的明文信息的信息摘要，然后，采用 CA的私钥对信息摘要进行加密，密文即签名;
* 客户端 Client 向服务器 Server 发出请求时，Server 返回证书文件;
* 客户端 Client 读取证书中的相关的明文信息，采用相同的散列函数计算得到信息摘要，然后，利用对应 CA的公钥解密签名数据，对比证书的信息摘要，如果一致，则可以确认证书的合法性，即服务器的公开密钥是值得信赖的。
* 客户端还会验证证书相关的域名信息、有效时间等信息; 客户端会内置信任CA的证书信息(包含公钥)，如果CA不被信任，则找不到对应 CA的证书，证书也会被判定非法。

https不是真正安全的

上边说道，https能避免传输的过程中，如果有人截获到数据包只能看到加密后的信息，但是防不了在服务端和客户端截取数据的人。服务器端自不必说，如果黑客都能取到服务器的数据了那你加不加密估计也没什么意义了，但客户端就不一样了，许多密码泄露都是在客户端泄露的。所以客户端密码保护很重要！显然https这点就做不到了。那么，就只有写程序的人自己定义加密方式了。

　　我们看百度的登录的过程

* 



　　我们可以很easily的看到，在登录之前，百度前端获取了一次公钥，用公钥加密后传输加密后的密文密码，这样做到了全流程加密传输。