

二

25 固若金汤的根本（下）：数字签名与证书

上一讲中我们学习了对称加密和非对称加密，以及两者结合起来的混合加密，实现了机密性。

但仅有机密性，离安全还差的很远。

黑客虽然拿不到会话密钥，无法破解密文，但可以通过窃听收集到足够多的密文，再尝试着修改、重组后发给网站。因为没有完整性保证，服务器只能“照单全收”，然后他就可以通过服务器的响应获取进一步的线索，最终就会破解出明文。

另外，黑客也可以伪造身份发布公钥。如果你拿到了假的公钥，混合加密就完全失效了。你以为自己是在和“某宝”通信，实际上网线的另一端却是黑客，银行卡号、密码等敏感信息就在“安全”的通信过程中被窃取了。

所以，在机密性的基础上还必须加上完整性、身份认证等特性，才能实现真正的安全。

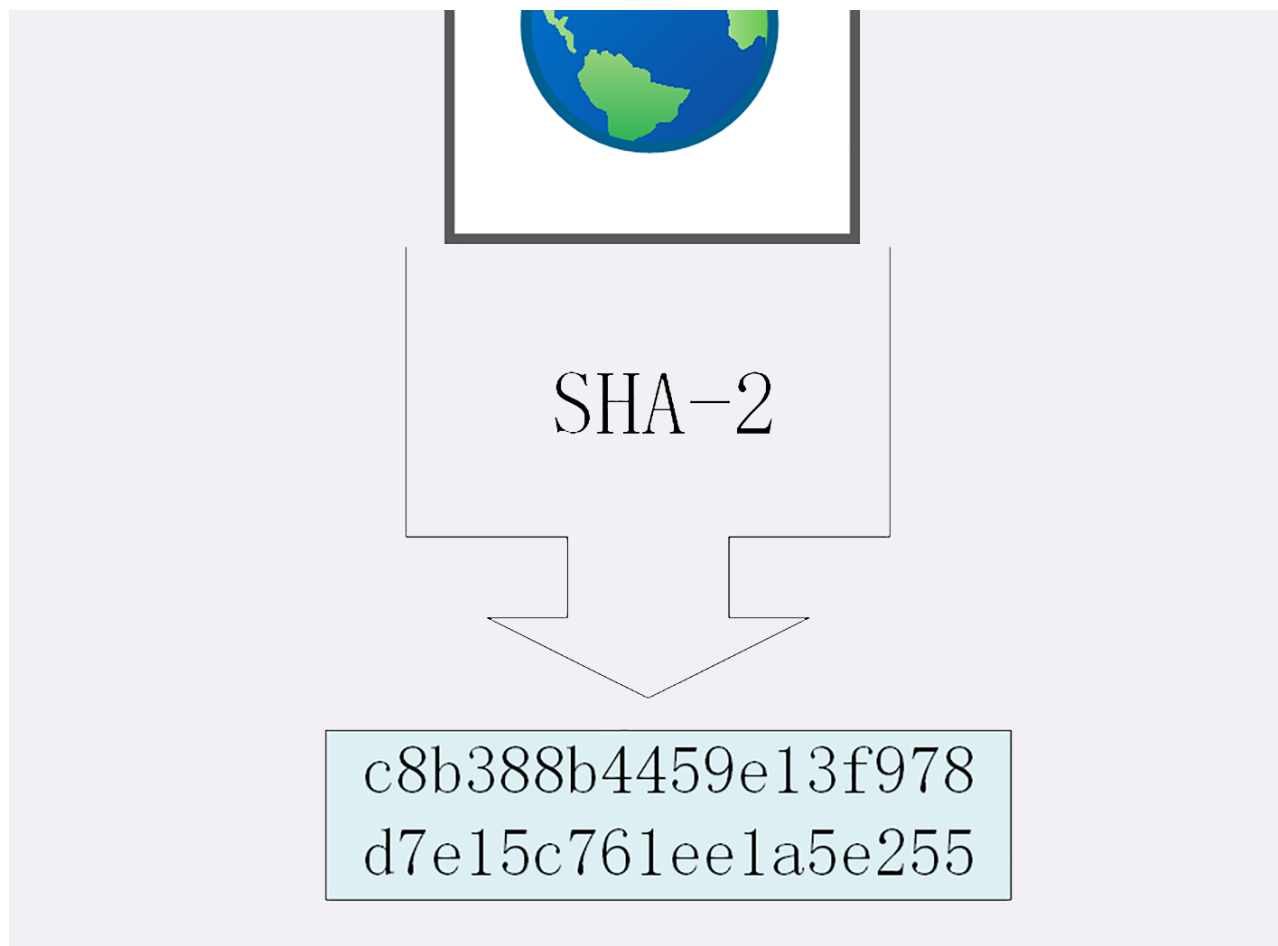
摘要算法

实现完整性的手段主要是**摘要算法**（Digest Algorithm），也就是常说的散列函数、哈希函数（Hash Function）。

你可以把摘要算法近似地理解成一种特殊的压缩算法，它能够把任意长度的数据“压缩”成固定长度、而且独一无二的“摘要”字符串，就好像是给这段数据生成了一个数字“指纹”。

换一个角度，也可以把摘要算法理解成特殊的“单向”加密算法，它只有算法，没有密钥，加密后的数据无法解密，不能从摘要逆推出原文。





摘要算法实际上是把数据从一个“大空间”映射到了“小空间”，所以就存在“冲突”（collision，也叫碰撞）的可能性，就如同现实中的指纹一样，可能会有两份不同的原文对应相同的摘要。好的摘要算法必须能够“抵抗冲突”，让这种可能性尽量地小。

因为摘要算法对输入具有“单向性”和“雪崩效应”，输入的微小不同会导致输出的剧烈变化，所以也被 TLS 用来生成伪随机数（PRF，pseudo random function）。

你一定在日常工作中听过、或者用过 MD5（Message-Digest 5）、SHA-1（Secure Hash Algorithm 1），它们就是最常用的两个摘要算法，能够生成 16 字节和 20 字节长度的数字摘要。但这两个算法的安全强度比较低，不够安全，在 TLS 里已经被禁止使用了。

目前 TLS 推荐使用的是 SHA-1 的后继者：SHA-2。

SHA-2 实际上是一系列摘要算法的统称，总共有 6 种，常用的有 SHA224、SHA256、SHA384，分别能够生成 28 字节、32 字节、48 字节的摘要。

你可以用实验环境的 URI“/25-1”来测试一下 TLS 里的各种摘要算法，包括 MD5、SHA-1 和 SHA-2。

<https://www.chrono.com/25-1?algo=md5>

<https://www.chrono.com/25-1?algo=sha1>
<https://www.chrono.com/25-1?algo=sha256>

完整性

摘要算法保证了“数字摘要”和原文是完全等价的。所以，我们只要在原文后附上它的摘要，就能够保证数据的完整性。

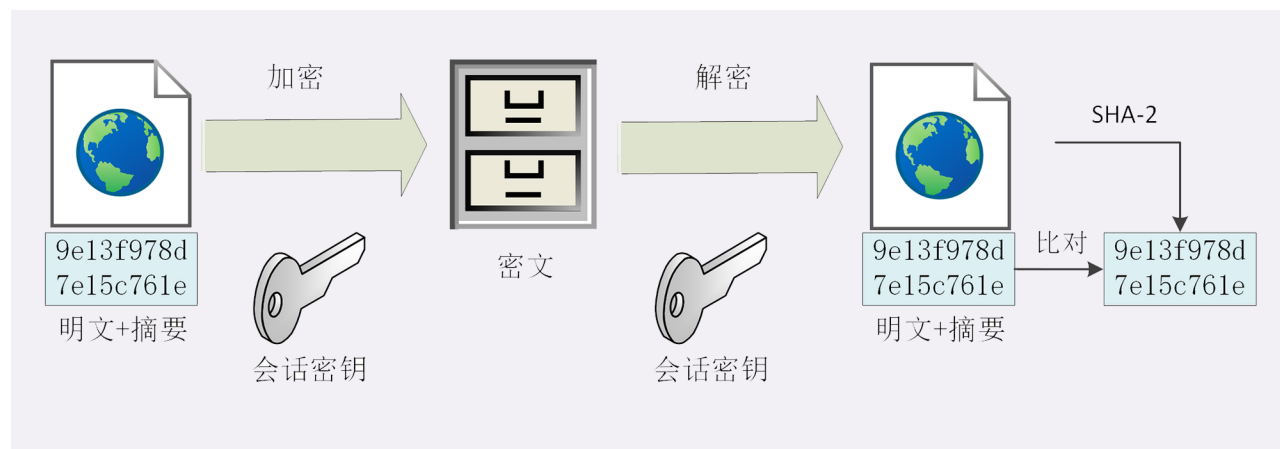
比如，你发了条消息：“转账 1000 元”，然后再加上一个 SHA-2 的摘要。网站收到后也计算一下消息的摘要，把这两份“指纹”做个对比，如果一致，就说明消息是完整可信的，没有被修改。

如果黑客在中间哪怕改动了一个标点符号，摘要也会完全不同，网站计算比对就会发现消息被篡改，是不可信的。

不过摘要算法不具有机密性，如果明文传输，那么黑客可以修改消息后把摘要也一起改了，网站还是鉴别不出完整性。

所以，真正的完整性必须要建立在机密性之上，在混合加密系统里用会话密钥加密消息和摘要，这样黑客无法得知明文，也就没有办法动手脚了。

这有个术语，叫哈希消息认证码（HMAC）。



数字签名

加密算法结合摘要算法，我们的通信过程可以说是比较安全了。但这里还有漏洞，就是通信的两个端点（endpoint）。

就像一开始所说的，黑客可以伪装成网站来窃取信息。而反过来，他也可以伪装成你，向网

站发送支付、转账等消息，网站没有办法确认你的身份，钱可能就这么被偷走了。

现实生活中，解决身份认证的手段是签名和印章，只要在纸上写下签名或者盖个章，就能够证明这份文件确实是由本人而不是其他人发出的。

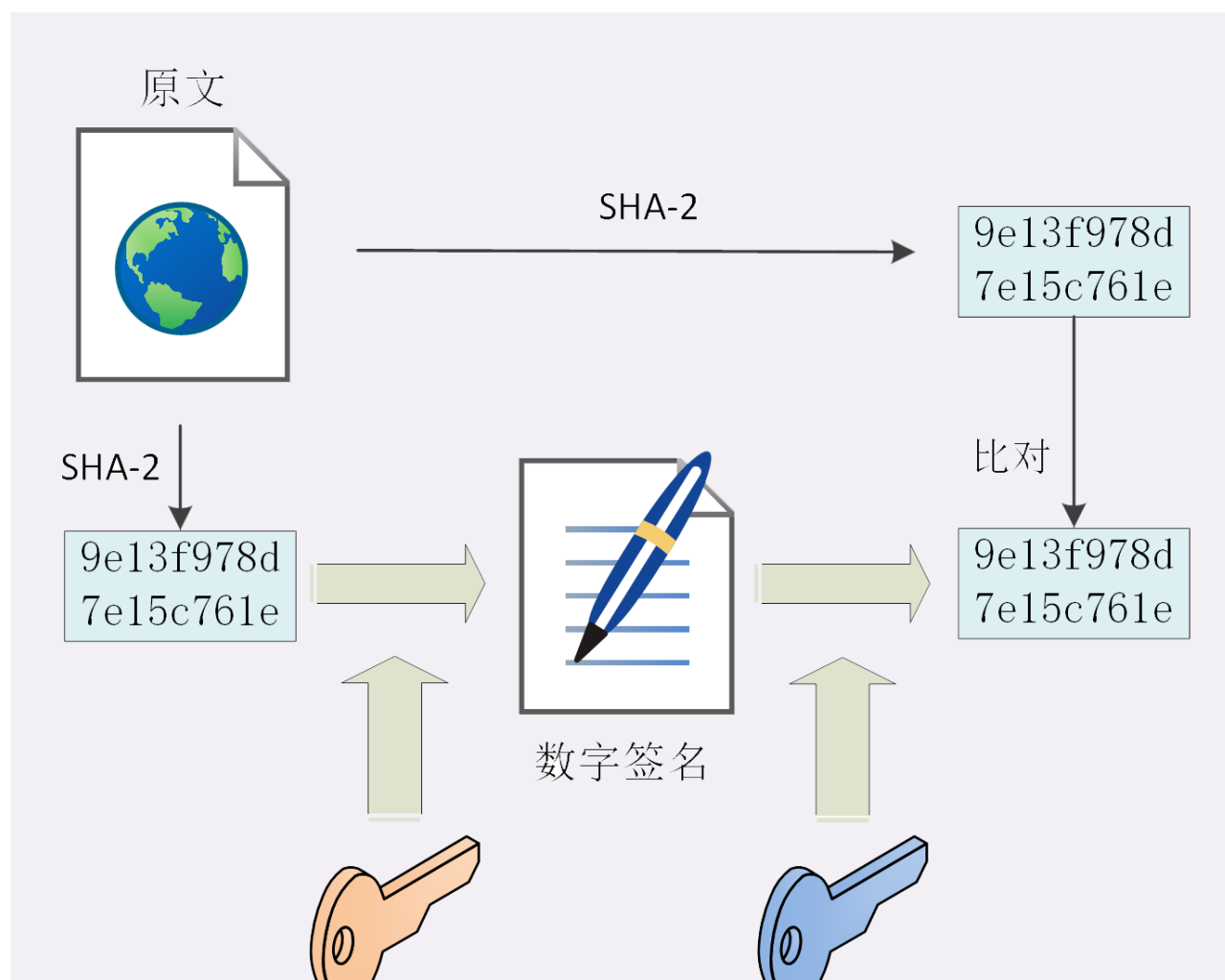
你回想一下之前的课程，在 TLS 里有什么东西和现实中的签名、印章很像，只能由本人持有，而其他任何人都不会有呢？只要用这个东西，就能够在数字世界里证明你的身份。


没错，这个东西就是非对称加密里的“**私钥**”，使用私钥再加上摘要算法，就能够实现“**数字签名**”，同时实现“身份认证”和“不可否认”。

数字签名的原理其实很简单，就是把公钥私钥的用法反过来，之前是公钥加密、私钥解密，现在是私钥加密、公钥解密。


但又因为非对称加密效率太低，所以私钥只加密原文的摘要，这样运算量就小的多，而且得到的数字签名也很小，方便保管和传输。

签名和公钥一样完全公开，任何人都可以获取。但这个签名只有用私钥对应的公钥才能解开，拿到摘要后，再比对原文验证完整性，就可以像签署文件一样证明消息确实是你发的。





私钥加密



公钥解密

刚才的这两个行为也有专用术语，叫做“**签名**”和“**验签**”。

只要你和网站互相交换公钥，就可以用“签名”和“验签”来确认消息的真实性，因为私钥保密，黑客不能伪造签名，就能够保证通信双方的身份。

比如，你用自己的私钥签名一个消息“我是小明”。网站收到后用你的公钥验签，确认身份没问题，于是也用它的私钥签名消息“我是某宝”。你收到后再用它的公钥验一下，也没问题，这样你和网站就都知道对方不是假冒的，后面就可以用混合加密进行安全通信了。

实验环境的 URI“/25-2”演示了 TLS 里的数字签名，它使用的是 RSA1024。

数字证书和 CA

到现在，综合使用对称加密、非对称加密和摘要算法，我们已经实现了安全的四大特性，是不是已经完美了呢？

不是的，这里还有一个“**公钥的信任**”问题。因为谁都可以发布公钥，我们还缺少防止黑客伪造公钥的手段，也就是说，怎么来判断这个公钥就是你或者某宝的公钥呢？

真是“按下葫芦又起了瓢”，安全还真是个麻烦事啊，“一环套一环”的。

我们可以用类似密钥交换的方法来解决公钥认证问题，用别的私钥来给公钥签名，显然，这又会陷入“无穷递归”。

但这次实在是“没招”了，要终结这个“死循环”，就必须引入“外力”，找一个公认的可信第三方，让它作为“信任的起点，递归的终点”，构建起公钥的信任链。

这个“第三方”就是我们常说的**CA**（Certificate Authority，证书认证机构）。它就像网络世界里的公安局、教育部、公证中心，具有极高的可信度，由它来给各个公钥签名，用自身的信誉来保证公钥无法伪造，是可信的。

CA 对公钥的签名认证也是有格式的，不是简单地把公钥绑定在持有者身份上就完事了，还要包含序列号、用途、颁发者、有效时间等等，把这些打成一个包再签名，完整地证明公钥关联的各种信息，形成“**数字证书**”（Certificate）。

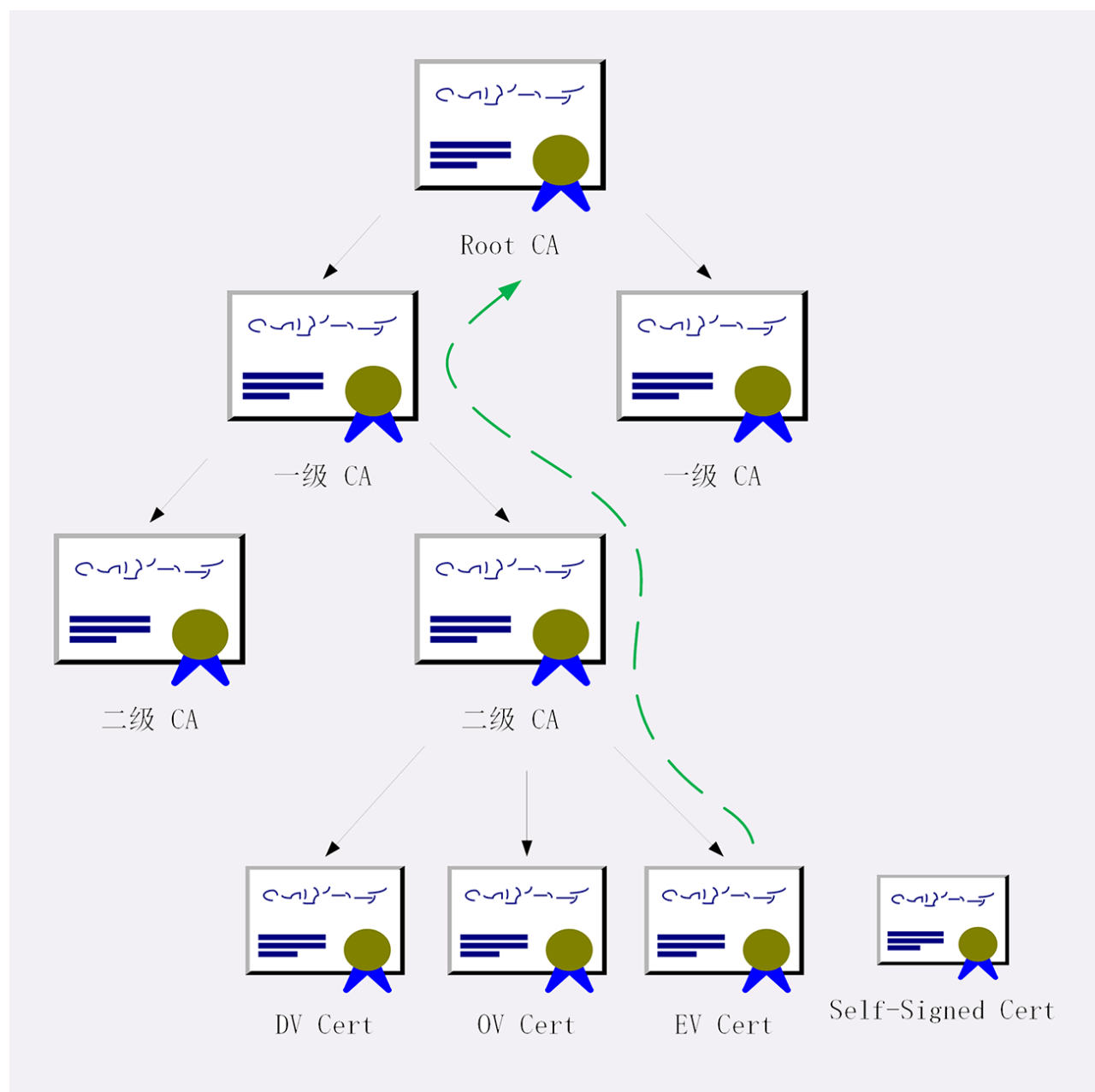
知名的 CA 全世界就那么几家，比如 DigiCert、VeriSign、Entrust、Let's Encrypt 等，它们

签发的证书分 DV、OV、EV 三种，区别在于可信程度。

DV 是最低的，只是域名级别的可信，背后是谁不知道。EV 是最高的，经过了法律和审计的严格核查，可以证明网站拥有者的身份（在浏览器地址栏会显示出公司的名字，例如 Apple、GitHub 的网站）。

不过，CA 怎么证明自己呢？

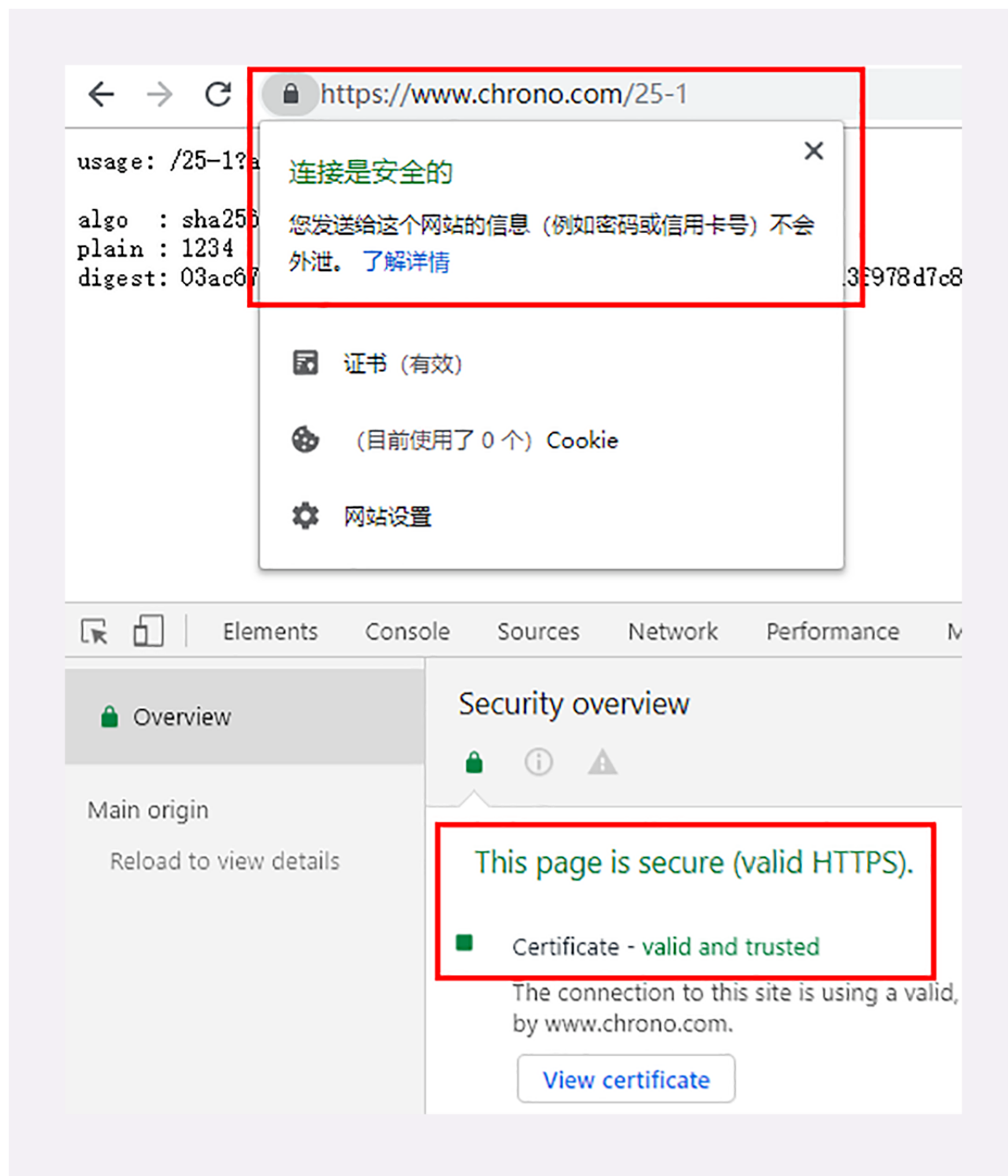
这还是信任链的问题。小一点的 CA 可以让大 CA 签名认证，但链条的最后，也就是**Root CA**，就只能自己证明自己了，这个就叫“**自签名证书**”（Self-Signed Certificate）或者“**根证书**”（Root Certificate）。你必须相信，否则整个证书信任链就走不下去了。



有了这个证书体系，操作系统和浏览器都内置了各大 CA 的根证书，上网的时候只要服务器

发过来它的证书，就可以验证证书里的签名，顺着证书链（Certificate Chain）一层层地验证，直到找到根证书，就能够确定证书是可信的，从而里面的公钥也是可信的。

我们的实验环境里使用的证书是“野路子”的自签名证书（在 Linux 上用 OpenSSL 命令行签发），肯定是不會被浏览器所信任的，所以用 Chrome 访问时就会显示成红色，标记为不安全。但你只要把它安装进系统的根证书存储区里，让它作为信任链的根，就不会再有危险警告。



证书体系的弱点

证书体系（PKI, Public Key Infrastructure）虽然是目前整个网络世界的安全基础设施，但绝对的安全是不存在的，它也有弱点，还是关键的“信任”二字。

如果 CA 失误或者被欺骗，签发了错误的证书，虽然证书是真的，可它代表的网站却是假的。

还有一种更危险的情况，CA 被黑客攻陷，或者 CA 有恶意，因为它（即根证书）是信任的源头，整个信任链里的所有证书也就都不可信了。

这两种事情并不是“耸人听闻”，都曾经实际出现过。所以，需要再给证书体系打上一些补丁。

针对第一种，开发出了 CRL（证书吊销列表，Certificate revocation list）和 OCSP（在线证书状态协议，Online Certificate Status Protocol），及时废止有问题的证书。

对于第二种，因为涉及的证书太多，就只能操作系统或者浏览器从根上“下狠手”了，撤销对 CA 的信任，列入“黑名单”，这样它颁发的所有证书就都会被认为是不安全的。

小结

今天我们学习了数字签名和证书、CA，是不是有种“盗梦空间”一层套一层的感觉？你可以在课后再去各大网站，结合它们“小锁头”里的信息来加深理解。

今天的内容可以简单概括为四点：

1. 摘要算法用来实现完整性，能够为数据生成独一无二的“指纹”，常用的算法是 SHA-2；
2. 数字签名是私钥对摘要的加密，可以由公钥解密后验证，实现身份认证和不可否认；
3. 公钥的分发需要使用数字证书，必须由 CA 的信任链来验证，否则就是不可信的；
4. 作为信任链的源头 CA 有时也会不可信，解决办法有 CRL、OCSP，还有终止信任。

课下作业

1. 为什么公钥能够建立信任链，用对称加密算法里的对称密钥行不行呢？
2. 假设有一个三级的证书体系（Root CA=> 一级 CA=> 二级 CA），你能详细解释一下证书信任链的验证过程吗？

欢迎你把自己的学习体会写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。



== 课外小贴士 ==

- 01 摘要算法除了用于 TLS 安全通信，还有很多其他的用途，比如散列表、数据校验、大文件比较等。
- 02 虽然 SHA-2 很安全，但出于“未雨绸缪”的考虑，又出现了 SHA-3，它也有 6 种算法，名字与 SHA-2 差不多，比如 SHA3-224、SHA3-256，目前还未纳入 TLS。
- 03 “账号 + 密码”也能够实现简单的的身份认证，但在安全通信未建立前使用很容易就会被窃取，所以在 TLS 里不能用。
- 04 Let's Encrypt 是著名的免费 CA，它只颁发 DV 证书，而且出于安全目的有效期只有 90 天，但可以用 Certbot 工具自动续订。

05 证书的格式遵循 X509 v3 标准，有两种编码方式，一种是二进制的 DER，另一种是 ASCII 码的 PEM，实验环境使用的是 PEM。

[上一页](#)[下一页](#)