

<!--

- @Description:
  - @Author: alphapenng
  - @Github:
  - @Date: 2023-03-29 14:00:47
  - @LastEditors: alphapenng
  - @LastEditTime: 2023-04-13 16:56:56
  - @FilePath: /balabala/content/private/DNS原理与应用.md
- >

# DNS 原理与应用

- **DNS 原理与应用**

- **DNS 是啥？有啥用？**
- **域名的结构是咋样滴？**
- **全球 DNS**
- **根域名**
- **一级域名**
- **二级域名**
- **三级域名**
- **DNS 缓存**
- **“域名解析” 是怎么实现的？**
- **域名服务器如何知道这些信息？**
  - **域名的缓存**
  - **缓存的同步**
  - **同步的周期**
- **什么是“域名劫持”？**
- **谁有“域名劫持”的企图？**
- **如何对付“域名劫持”？**
- **啥是“域名污染”？**
- **谁有“域名污染”的企图？**

- **怎么对付“域名污染/域名欺骗”？**

- **DNS over TLS**

- **历史**
    - **协议栈**
    - **安全性原理**

- **DNS over HTTPS**

- **协议栈**
    - **安全性原理**

- **自建 DNS 服务器**

## DNS 是啥？有啥用？

DNS 是 “Domain Name System” 的缩写，直译过来就是 “域名系统”。

咱们每天打交道的这个互联网，其底层的基石是 “IP”。IP 是 “Internet Protocol” 的缩写，中文就 “互联网协议”（光看名字就知道这玩意儿很重要）。咱们日常用的那些互联网软件（浏览器、聊天工具、下载工具、等等）在工作时，必须依靠【IP 地址】才能进行网络数据传输。

“IP 地址” 是设计给软件用滴 —— 虽然软件很容易处理，但对于人类而言，却很难记忆。于是，后来又发明了 DNS。有了 DNS，人类就不需要记住长长的一串 IP 地址，而只需记住 “域名”（域名通常更短，也更具有可读性）。

比如你上网的时候，只需在地址栏输入网站的 “域名”，而不用输入网站的 “IP 地址”。然后电脑系统会利用 DNS 来把 “域名” 翻译成 “IP 地址”。这个翻译的过程术语叫 “域名解析 / DNS 解析”。

## 域名的结构是咋样滴？

互联网上的域名按照【树形层次结构】来组织。不懂得啥是 “树形结构” 的同学，可以对照一下电脑硬盘上的目录结构。域名的结构和目录结构很类似，目录结构是用 “斜杠” 作分隔符，而域名是用小数点作分隔符。两者的主要区别在于：目录结构名称的形式是从【左到右】（上级在左，下级在右），而域名是从【右到左】（上级在右，下级在左）。

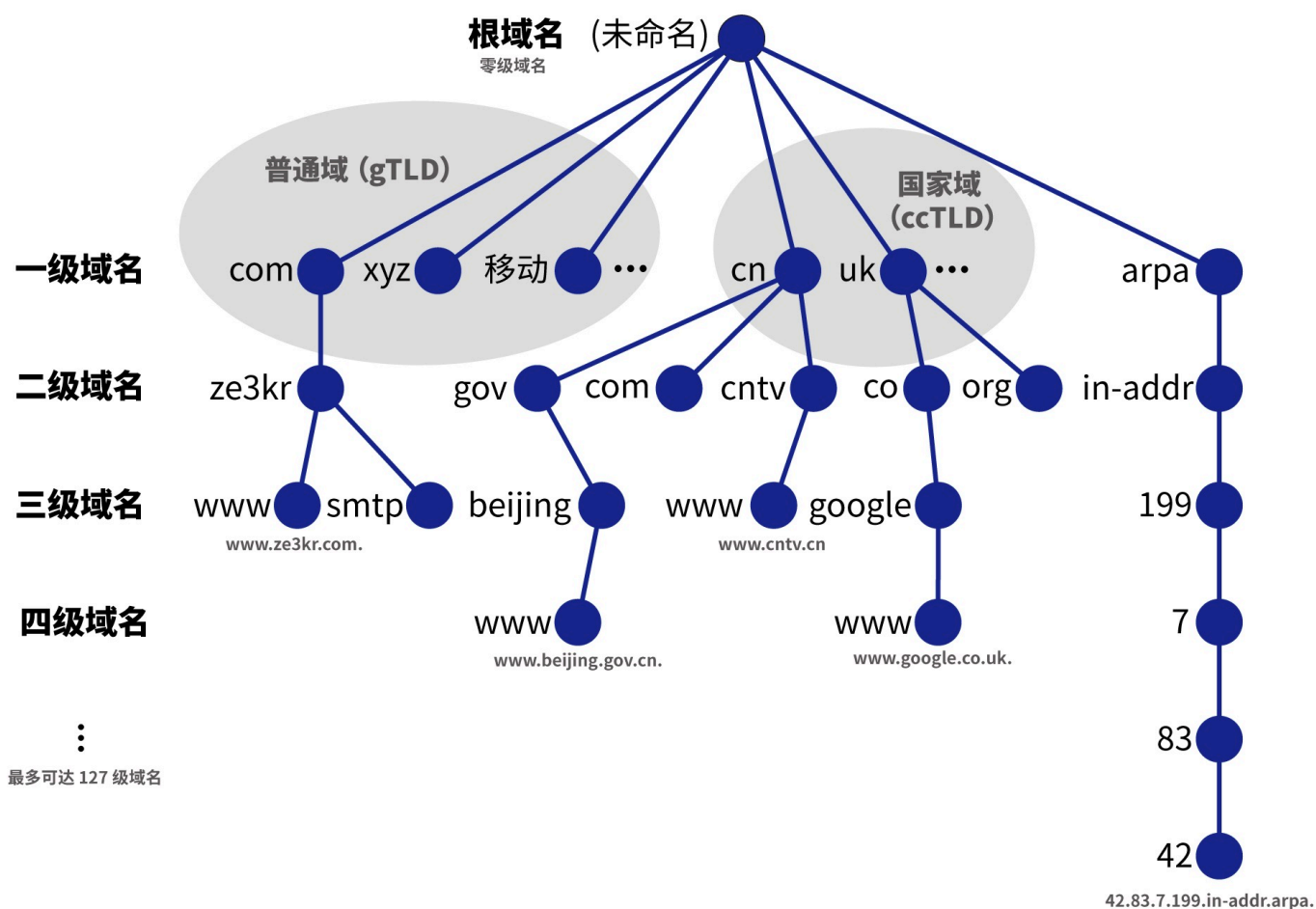
例如：**www.baidu.com** 的上级域名是 .baidu.com

.baidu.com 的上级域名是 .com

这里的 .com 就被称为顶级域名（Top-Level Domain，简称 TLD），跟 .com 类似的那些 .net、.org、.gov 也都是顶级域名。还有那些以国家 / 地区的代码命名的（比如：.cn、.hk、.jp 等等）也是顶级域名。

## 全球 DNS

在全球 DNS 中，一个完整域名通常包含多级，比如 example.com. 就是个二级域名，**www.example.com** 就是个三级域名。通常我们常见到的域名都是完整的域名。



一级域名被分为以下三个部分：

1. **普通域 (gTLD)**：通常是三个字节或三个字节以上的域名后缀，或者是 Unicode 字符的后缀。这些域名分配给机构管理。
2. **国家域 (ccTLD)**：所有两个字节的域名都是国家代码，这些域名分配给国家管理。不少国家域都开放了注册，不过有的国家域仅允许当前国家的人去注册。
3. **arpa 域**：用于将 IP 地址转换为对应的域名的根域。

我们通常所见到的域名都是普通域和国家域，而 arpa 域用作 IP 到域名的反向解析。在本地 DNS 中，只存在域名对应 IP 这种映射关系。然而，在全球 DNS 中，有着更多的资源记录种类 (RR)，不只是域名对应 IP 的关系，下面将分别介绍一些最基本的资源记录种类：

- **A 记录**：定义了一个 IP 地址。（AAAA 记录则是定义一个 IPv6 地址）
- **NS 记录**：域名服务器记录，说明一个域名下的授权域名服务器记录。内容必须是一个域名。

# 根域名

先从根域名开始，未命名根也可以作为.。你在接下来的部分所看到的很多域名都以.结尾，以.结尾的域名是特指的是根域名下的完整域名，然而不以.结尾的域名大都也是完整域名，实际使用时域名末尾的.也常常省略。在本文中，我使用 dig 这一个常用的 DNS 软件来进行查询，我的电脑也已经连接到了互联网。假设目前这个计算机能与互联网上的 IP 通信，但是完全没有 DNS 服务器。此时你需要知道根 DNS 服务器，以便自己获取某个域名的 IP 地址。根 DNS 服务器列表可以在<https://www.internic.net/domain/named.root>下载到。文件内容如下：

```
1 ;      This file holds the information on root name servers needed to
2 ;      initialize cache of Internet domain name servers
3 ;      (e.g. reference this file in the "cache . <file>"
4 ;      configuration file of BIND domain name servers).
5 ;
6 ;      This file is made available by InterNIC
7 ;      under anonymous FTP as
8 ;          file                /domain/named.cache
9 ;          on server            FTP.INTERNIC.NET
10 ;      -OR-                    RS.INTERNIC.NET
11 ;
12 ;      last update:      October 20, 2016
13 ;      related version of root zone:  2016102001
14 ;
15 ; formerly NS.INTERNIC.NET
16 ;
17 .                3600000      NS      A.ROOT-SERVERS.NET.
18 A.ROOT-SERVERS.NET.  3600000      A      198.41.0.4
19 A.ROOT-SERVERS.NET.  3600000      AAAA   2001:503:ba3e::2:30
20 ;
21 ; FORMERLY NS1.ISI.EDU
22 ;
23 .                3600000      NS      B.ROOT-SERVERS.NET.
24 B.ROOT-SERVERS.NET.  3600000      A      192.228.79.201
25 B.ROOT-SERVERS.NET.  3600000      AAAA   2001:500:84::b
26 ;
27 ; FORMERLY C.PSI.NET
28 ;
29 .                3600000      NS      C.ROOT-SERVERS.NET.
30 C.ROOT-SERVERS.NET.  3600000      A      192.33.4.12
31 C.ROOT-SERVERS.NET.  3600000      AAAA   2001:500:2::c
32 ;
33 ; FORMERLY TERP.UMD.EDU
34 ;
```

```

35 . 3600000 NS D.ROOT-SERVERS.NET.
36 D.ROOT-SERVERS.NET. 3600000 A 199.7.91.13
37 D.ROOT-SERVERS.NET. 3600000 AAAA 2001:500:2d::d
38 ;
39 ; FORMERLY NS.NASA.GOV
40 ;
41 . 3600000 NS E.ROOT-SERVERS.NET.
42 E.ROOT-SERVERS.NET. 3600000 A 192.203.230.10
43 E.ROOT-SERVERS.NET. 3600000 AAAA 2001:500:a8::e
44 ;
45 ; FORMERLY NS.ISC.ORG
46 ;
47 . 3600000 NS F.ROOT-SERVERS.NET.
48 F.ROOT-SERVERS.NET. 3600000 A 192.5.5.241
49 F.ROOT-SERVERS.NET. 3600000 AAAA 2001:500:2f::f
50 ;
51 ; FORMERLY NS.NIC.DDN.MIL
52 ;
53 . 3600000 NS G.ROOT-SERVERS.NET.
54 G.ROOT-SERVERS.NET. 3600000 A 192.112.36.4
55 G.ROOT-SERVERS.NET. 3600000 AAAA 2001:500:12::d0d
56 ;
57 ; FORMERLY AOS.ARL.ARMY.MIL
58 ;
59 . 3600000 NS H.ROOT-SERVERS.NET.
60 H.ROOT-SERVERS.NET. 3600000 A 198.97.190.53
61 H.ROOT-SERVERS.NET. 3600000 AAAA 2001:500:1::53
62 ;
63 ; FORMERLY NIC.NORDU.NET
64 ;
65 . 3600000 NS I.ROOT-SERVERS.NET.
66 I.ROOT-SERVERS.NET. 3600000 A 192.36.148.17
67 I.ROOT-SERVERS.NET. 3600000 AAAA 2001:7fe::53
68 ;
69 ; OPERATED BY VERISIGN, INC.
70 ;
71 . 3600000 NS J.ROOT-SERVERS.NET.
72 J.ROOT-SERVERS.NET. 3600000 A 192.58.128.30
73 J.ROOT-SERVERS.NET. 3600000 AAAA 2001:503:c27::2:30
74 ;
75 ; OPERATED BY RIPE NCC
76 ;
77 . 3600000 NS K.ROOT-SERVERS.NET.
78 K.ROOT-SERVERS.NET. 3600000 A 193.0.14.129

```

```

79 K.ROOT-SERVERS.NET.      3600000      AAAA  2001:7fd::1
80 ;
81 ; OPERATED BY ICANN
82 ;
83 .              3600000      NS    L.ROOT-SERVERS.NET.
84 L.ROOT-SERVERS.NET.      3600000      A      199.7.83.42
85 L.ROOT-SERVERS.NET.      3600000      AAAA  2001:500:9f::42
86 ;
87 ; OPERATED BY WIDE
88 ;
89 .              3600000      NS    M.ROOT-SERVERS.NET.
90 M.ROOT-SERVERS.NET.      3600000      A      202.12.27.33
91 M.ROOT-SERVERS.NET.      3600000      AAAA  2001:dc3::35
92 ; End of file

```

这个文件中每一行分为 4 列，分别是完整域名、资源类型、生存时间（TTL，也就是可以缓存的时间）以及资源数据。通过这个文件就可以知道根域名所对应的 13 个根域名服务器的完整域名，还能知道这 13 个完整域名所对应的 IP 地址。这是因为 NS 记录只能设置为完整域名，所以为了告知 NS 所对应的 IP，还需要额外的数据去说明这 13 个完整域名对应的 IP（这些额外的数据叫做 Glue 记录）。这 13 个根域名服务器都是独立的且冗余的（但是所返回的内容应该是相同的），这样当其中的某个或某些服务器发生故障时，不会影响到解析。一旦无法解析根域名，那么所有的域名都将无法访问。全球 13 个根服务器节点集群分布在美国、欧洲、日本。截至 2021 年 7 月，中国大陆共有 F、I、J、K、L 这 5 个根域的 21 台 DNS 镜像在提供服务。在主服务器断联后亦可继续互联网 DNS 解析服务，但其并未有根域名服务器的资格。

DNS 镜像根服务器（Root Name Server Mirror）是一组在互联网中分布的镜像 DNS 根服务器。它们的作用是分担 DNS 根服务器的负载并提高 DNS 解析的性能和可靠性。DNS 镜像根服务器并不直接参与 DNS 查询，它们只是从 DNS 根服务器复制并缓存了一份 DNS 根服务器的信息，以便更快、更准确地响应 DNS 查询请求。

区别：

1. 数量：DNS 根服务器共有 13 个，DNS 镜像根服务器的数量比 DNS 根服务器要多得多。
2. 功能：DNS 根服务器直接参与 DNS 查询，提供域名到 IP 地址的转换服务。DNS 镜像根服务器仅仅缓存、镜像 DNS 根服务器的信息，提高 DNS 解析的速度和可靠性。
3. 位置：DNS 根服务器分布在全球不同的地理位置上，DNS 镜像根服务器也分布在全球不同的地理位置上，但是与 DNS 根服务器可能会有所不同。
4. 管理：DNS 根服务器由 ICANN（互联网名称与数字地址分配机构）组织负责管理，DNS 镜像根服务器可能由不同的组织或机构管理。

| 字母   | IPv4 地址                             | IPv6 地址                               | 自治系统编号 <sup>[2]</sup>  | 曾用名              | 运营单位                 | 设置地点<br>#数量（全球 / 地区） <sup>[3]</sup>                               | 软件  |
|--|-------------------------------------|---------------------------------------|--|------------------|----------------------|---|---|
| <a href="#">A</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 198.41.0.4                          | 2001:503:ba3e::2:30                   | AS26415, AS19836, <sup>[2][注 1]</sup> AS36619, AS36620, AS36622, AS36625, AS36631, AS64820 <sup>[注 2][4]</sup> | ns.internic.net  | 威瑞信                  | 以 <b>任播</b> 技术设置于多处<br>5/0  | NSD、威瑞信<br>ATLAS                                      |
| <a href="#">B</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 199.9.14.201 <sup>[注 3][5][6]</sup> | 2001:500:200::b <sup>[7]</sup>        | AS394353 <sup>[8]</sup>  | ns1.isi.edu      | 美国南加州大学信息学研究所        | 以 <b>任播</b> 技术设置于多处<br>2/0  | BIND  |
| <a href="#">C</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 192.33.4.12                         | 2001:500:2::c                         | AS2149 <sup>[2][9]</sup>   | c.psi.net        | Cogent 通信            | 以 <b>任播</b> 技术设置于多处<br>8/0  | BIND  |
| <a href="#">D</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 199.7.91.13 <sup>[注 4][10]</sup>    | 2001:500:2d::d                        | AS10886 <sup>[2][11]</sup>   | terp.umd.edu     | 美国马里兰大学学院市分校         | 以 <b>任播</b> 技术设置于多处<br>50/67                                      | BIND  |
| <a href="#">E</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 192.203.230.10                      | 2001:500:a8::e                        | AS21556 <sup>[2][12]</sup>   | ns.nasa.gov      | 美国国家航空航天局<br>埃姆斯研究中心 | 以 <b>任播</b> 技术设置于多处<br>125/141                                    | BIND、NSD  |
| <a href="#">F</a>  | 192.5.5.241                         | 2001:500:2f::f                        | AS3557, <sup>[2][13]</sup> AS1280, AS30132 <sup>[13]</sup>   | ns.isc.org       | 互联网系统协会              | 以 <b>任播</b> 技术设置于多处<br>57/0                                       | BIND <sup>[14]</sup>                                  |
| <a href="#">G</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）<sup>[注 5]</sup></small> | 192.112.36.4 <sup>[注 6]</sup>       | 2001:500:12::d0d <sup>[注 7]</sup>     | AS5927 <sup>[2][15]</sup>  | ns.nic.ddn.mil   | 美国国防信息系统局            | 以 <b>任播</b> 技术设置于多处<br>6/0  | BIND  |
| <a href="#">H</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 198.97.190.53 <sup>[注 8][16]</sup>  | 2001:500:1::53 <sup>[注 9][17]</sup>   | AS1508 <sup>[17][注 10][18]</sup>   | aos.arl.army.mil | 美国陆军研发实验室            | <a href="#">美国马里兰州阿伯丁试验场</a> 以及 <a href="#">加利福尼亚州圣地亚哥</a><br>2/0 | NSD   |
| <a href="#">I</a>  | 192.36.148.17                       | 2001:7fe::53                          | AS29216 <sup>[2][19]</sup>   | nic.nordu.net    | Netnod               | 以 <b>任播</b> 技术设置于多处<br>58/0                                       | BIND  |
| <a href="#">J</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 192.58.128.30 <sup>[注 11]</sup>     | 2001:503:c27::2:30                    | AS26415, <sup>[2][20]</sup> AS36626, AS36628, AS36632 <sup>[20]</sup>  | 不适用              | 威瑞信                  | 以 <b>任播</b> 技术设置于多处<br>61/13                                      | NSD、威瑞信<br>ATLAS                                      |
| <a href="#">K</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 193.0.14.129                        | 2001:7fd::1                           | AS25152 <sup>[2][21][22]</sup>   | 不适用              | 欧洲 IP 资源网络协调中心       | 以 <b>任播</b> 技术设置于多处<br>5/23                                       | BIND、NSD、<br><a href="#">Knot DNS</a> <sup>[23]</sup> |
| <a href="#">L</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 199.7.83.42 <sup>[注 12][24]</sup>   | 2001:500:9f::42 <sup>[注 13][25]</sup> | AS20144 <sup>[2][26][27]</sup>   | 不适用              | ICANN                | 以 <b>任播</b> 技术设置于多处<br>161/0                                      | NSD、 <a href="#">Knot DNS</a> <sup>[28]</sup>         |
| <a href="#">M</a> <small>（<a href="#">页面存档备份</a>，存于<a href="#">互联网档案馆</a>）</small>                 | 202.12.27.33                        | 2001:dc3::35                          | AS7500 <sup>[2][29][30]</sup>  | 不适用              | 日本 WIDE 项目           | 以 <b>任播</b> 技术设置于多处<br>6/1  | BIND  |

我们假设你已经通过某种方法成功获取到了这个文件，那么下一步，就是使用这里的服务器对根域名以及一级域名进行解析。**对根域名的解析实际上是不必要的，但是我们还是对其进行解析以便进一步分析，获得在互联网上最新、最全的数据。**在根域名上的记录，以从根域名服务器中所解析其根域名的数据为准，而不是刚才的那个文件中的内容。刚才的文件内容只是告知根域名服务器的列表，也就是只有 NS 记录和 NS 记录对应的完整域名的 IP 记录，而不是根域名下的所有记录。我们先向 198.41.0.4 这个 IP 发送查询根域名的所有记录，any 代表显示任何类型的记录。

```
1 dig @198.41.0.4 . any
```

**SOA 记录：**指定有关 *DNS* 区域的权威性信息，包含主要名称服务器、域名管理员的电邮地址、域名的流水式编号、和几个有关刷新区域的定时器。

**DNS 区域：**对于根域名来说，DNS 区域就是空的，也就是说它负责这互联网下所有的域名。而对于 com 域，DNS 区域就是 com.，管理着 com. 本身及其子域名的记录。

此处的 ADDITIONAL SECTION 其实就包含了 Glue 记录。



## 一级域名

根域名自身的 DNS 服务器除了被用于解析根自身之外，还用于解析所有在互联网上的一级域名。你会发现，几乎所有的 DNS 服务器，无论是否是根 DNS 服务器，都会解析其自身以及其下级域名。

```
1 dig @198.41.0.4 com any
```

## 二级域名

和解析一级域名 com. 时类似，继续使用 com. 的域名服务器解析 baidu.com.。

```
1 dig @192.5.6.30 baidu.com any
```

## 三级域名

```
1 dig @220.181.33.31 www.baidu.com any
```

## DNS 缓存

通常，凡是解析过的记录，都会被解析服务器、路由器、客户端、软件缓存。这样可以大大减少请求次数。凡是被缓存的记录，其在 TTL 规定的时间内都不会再次去解析，而是直接从高速缓存中读取。

## “域名解析” 是怎么实现的？

如果你曾经配置过电脑的网卡，应该记得上面除了有 IP 地址、掩码等设置，还有一项设置是 “DNS 服务器 / 域名服务器”。这项设置就是用来帮助你的电脑进行域名解析的。你可以把这个 “DNS 服务器” 想象成 114 查号台。每当电脑需要翻译某个域名，就找这个域名服务器查询，然后域名服务器会告诉你的电脑，要查询的域名对应的 IP 地址是啥。

下面简单说一下，你的电脑进行域名解析的过程。

为了叙述方便，以百度网址为例。当你在浏览器的地址栏中输入 <https://www.baidu.com/>，然后敲回车，这时候电脑软件会进行如下系列事情：

1. 首先根据输入的网址，提取出域名（在本例中，也就是 [www.baidu.com](https://www.baidu.com/)）
2. 如果你在系统中配置了 Hosts 文件，那么电脑会先查询 Hosts 文件，看这个 [www.baidu.com](https://www.baidu.com/) 否已经在 Hosts (Windows hosts 文件位置为 "C:\Windows\System32\drivers\etc"，Linux hosts 文件位置为 "/etc/hosts") 里面有了对应的记录。如果有，直接就可以拿到该记录中的 IP 地址，过程就结束了。



3. 如果 Hosts 里面没有这个别名，那么电脑会看你有没有设置域名服务器（DNS 服务器）。如果你的系统没有设置域名服务器，那电脑就没辙了，浏览器直接会报错，说网站的域名无法解析。过程就结束了。
4. 如果你设置过“域名服务器”，那么电脑会向这个域名服务器发送一个域名查询（DNS query）的请求，然后等候域名服务器的回应。
5. 如果域名服务器始终没有回应（比如域名服务器挂了，或域名服务器的 IP 填错了），那么电脑还是没辙（浏览器会报错）。
6. 如果域名服务器回应了，那么你的电脑就可以根据域名服务器的应答信息，得到该域名的 IP 地址。之后浏览器就会向这个 IP 地址对应的 Web 端口发送 HTTP 请求。

通常情况下，电脑拿到的（DNS 服务器）应答信息是正确的——也就是说，应答中的 IP 地址确实对应那个域名——这种情况下，你的网络软件就可以正常工作了。

但是电脑拿到的 DNS 应答有可能是【错的】。为啥会这样捏，之后还会给大家介绍一下“域名劫持”和“域名污染”。

## 域名服务器如何知道这些信息？

### 域名的缓存

大伙儿平时使用的域名服务器，技术术语叫“递归域名服务器”。“递归服务器”是面向普通网友的。刚才介绍“域名解析”的时候提到的服务器就是“递归服务器”。

“递归服务器”的内部通常会有一个“DNS 记录的缓存”——这个缓存是为了提高查询效率的。当某台电脑向递归服务器发起域名查询时，递归服务器首先看自己的缓存中有没有该域名的记录，如果有，直接就回复该记录给查询的电脑。

万一对方想要查询的域名没找到，咋办捏？这时候就要进行缓存的同步。

### 缓存的同步

下面就拿百度的域名为例，说说这种情况的处理流程。

1. 当查询 **www.baidu.com** 这个域名，“递归服务器”发现自己的缓存中没有
  2. “递归服务器”会先去找“根域名服务器”帮忙，“根服务器”会告诉“递归服务器”说：这个域名属于 com 这个分支之下，你去找 com 这个域名的“权威服务器”，这个权威服务器的 IP 地址是 xxx。
  3. 然后“递归服务器”根据拿到的这个 xxx 地址，又去找“com 域名的权威服务器”。“com 域名的权威服务器”告诉它：你应该去找“baidu.com 域名的权威服务器”，这个权威服务器的 IP 地址是 yyy
  4. 然后“递归服务器”又屁颠屁颠地去找“baidu.com 域名的权威服务器”。这时候“baidu.com 域名的权威服务器”才会告诉它，**www.baidu.com** 这个域名的 IP 地址到底是多少。
- 大伙儿看到没有？整个过程如同“踢皮球”，效率是很低的。所以俺前面提到，“递归域名服务器”必须得有一个缓存，以此来优化效率（不用每次查询都来一次“踢皮球”）。

## 同步的周期

说完了“域名的同步”，顺便提一下“同步的周期”。

因为互联网上的域名信息是有可能发生变化的。比如增加了某个新域名，注销了某个旧域名，或者某个域名对应的 IP 地址变了。所以，“递归服务器”上保留的缓存中，每一条域名记录都有一个生命周期（可能是几分钟，也可能是几小时）。如果某条记录的生命周期过了，就会被删除，然后重新同步。

## 什么是“域名劫持”？

刚才说了，域名服务器上都会保存一大堆的域名记录（每条记录包含“域名”和“IP 地址”）。当收到域名查询的时候，域名服务器会从这堆记录中找到对方想要的，然后回应给对方。

如果域名服务器上的某条记录被【人为修改】了（改成错的），那么一旦要查询这条记录，得到的就是错误的结果。这种情况称之为“域名劫持”。

## 谁有“域名劫持”的企图？

“域名劫持”通常是电信运营商（ISP）干的好事儿。很多宽带用户用的域名服务器就是 ISP 提供给你的。而 ISP 也是很奇葩的——经常耍流氓。

## 如何对付“域名劫持”？

刚才说了，“域名劫持”的根源在于：域名服务器上的记录被人给改了。要对付这种耍流氓，最直接的办法就是不要使用这种流氓 ISP 提供的域名服务器，改用那些比较靠谱的或者是自建 DNS 服务器。

## 啥是“域名污染”？

“域名污染”这个词还有其它几个别名，分别是“域名欺骗”、“域名缓存投毒”（全称叫：“DNS cache poisoning”）。今后看到这几个别名，就是指同一个意思。

“域名污染”的原理，简单说来是这样滴：当你的电脑向域名服务器发送了“域名查询”的请求，然后域名服务器把回应发送给你的电脑，这里存在一个【时间差】。如果某个攻击者能够在域名服务器的“DNS 应答”还没有到达你的电脑之前，先伪造一个错误的“DNS 应答”发给你电脑。那么你的电脑收到的就是错误的信息，并得到一个错误的 IP 地址。

## 谁有“域名污染”的企图？

从技术上讲，只要攻击者能够位于“你”和“域名服务器”的传输线路中间，那么攻击者就有机会搞“域名污染”。能够做到这点的，可能是一个黑客 / 骇客，也可能是 ISP。

用了这招之后，那些不明所以的网友只要是通过【域名的方式】访问该网站，他们的电脑进行 DNS 查询之后，多半会得到错误的结果（也就是说，查到的 IP 地址【是假的】）；既然拿到假 IP，当然就无法打开这个网站的页面啦。

# 怎么对付“域名污染/域名欺骗”？

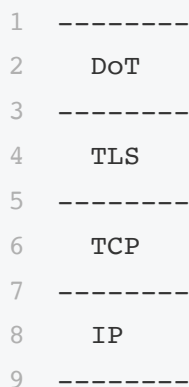
## DNS over TLS

“DNS over TLS” 有时也被简称为【DoT】。本文以下部分用 DoT 来称呼之。

### 历史

DoT 标准规范参见 RFC 7858 和 RFC 8310。（RFC 全称 Request for Comments，是 IETF（Internet Engineering Task Force，互联网工程任务组）组织记录互联网标准、协议、过程和民间惯例的文档。RFC 不是法律或严格的标准，而是一组规范性文件，提供了该组织工作过程的定义，包括协议、程序和原则。）从时间上看，RFC7858 是 2016 年发布的，RFC8310 是 2018 年发布的；显然，这个协议还是比较新的。

### 协议栈



### 安全性原理

#### • 几个术语——HTTPS、SSL、TLS

##### 1. “HTTP” 是干嘛用滴？

首先，HTTP 是一个网络协议，是专门用来帮你传输 Web 内容滴。比如你访问百度的主页，浏览器地址栏会出现如下的网址：<http://www.baidu.com>。

##### 2. “SSL/TLS” 是干嘛用滴？

SSL 是 “Secure Sockets Layer” 的缩写，中文叫做 “安全套接层”。它是在上世纪 90 年代中期，由网景公司设计的。

为啥要发明 SSL 这个协议捏？因为原先互联网上使用的 HTTP 协议是明文的，存在很多缺点——比如传输内容会被偷窥（嗅探）和篡改。发明 SSL 协议，就是为了解决这些问题。

到了 1999 年，SSL 因为应用广泛，已经成为互联网上的事实标准。IETF 就在那年把 SSL 标准化。标准化之后的名称改为 TLS（是 “Transport Layer Security” 的缩写），中文叫做 “传输层安全协议”。

很多相关的文章都把这两者并列称呼（SSL/TLS），因为这两者可以视作同一个东西的不同阶段。

### 3. “HTTPS” 是啥意思？

解释完 HTTP 和 SSL/TLS，现在就可以来解释 HTTPS 啦。咱们通常所说的 HTTPS 协议，说白了就是“HTTP 协议”和“SSL/TLS 协议”的组合。你可以把 HTTPS 大致理解为——“HTTP over SSL”或“HTTP over TLS”（反正 SSL 和 TLS 差不多，你可以把这俩当作同义词）。

## ● “对称加密”和“非对称加密”的概念

### 1. 啥是“加密”和“解密”？

通俗而言，你可以把“加密”和“解密”理解为某种【互逆的】数学运算。就好比“加法和减法”互为逆运算、“乘法和除法”互为逆运算。

“加密”的过程，就是把“明文”变成“密文”的过程；反之，“解密”的过程，就是把“密文”变为“明文”。在这两个过程中，都需要一个关键的东东——叫做“密钥”——来参与数学运算。

### 2. 啥是“对称加密”？

所谓的“对称加密技术”，意思就是说：“加密”和“解密”使用【相同的】密钥。这个比较好理解。

### 3. 啥是“非对称加密”？

所谓的“非对称加密技术”，意思就是说：“加密”和“解密”使用【不同的】密钥。这玩意儿比较难理解，也比较难想到。当年“非对称加密”的发明，还被誉为“密码学”历史上的一次革命。

### 4. 各自有啥优缺点？

看完刚才的定义，很显然：（从功能角度而言）“非对称加密”能干的事情比“对称加密”要多。这是“非对称加密”的优点。但是“非对称加密”的实现，通常需要涉及到“复杂数学问题”。所以，“非对称加密”的性能通常要差很多（相对于“对称加密”而言）。

这两者的优缺点，也影响到了 SSL 协议的设计。

## ● CA 证书的原理及用途

### 1. 先说一个通俗的例子

考虑到证书体系的相关知识比较枯燥、晦涩。俺先拿一个通俗的例子来说事儿。




普通的介绍信

假设 A 公司的张三先生要到 B 公司去拜访，但是 B 公司的所有人都不认识他，他咋办捏？常用的办法是带公司开的一张介绍信，在信中说：兹有张三先生前往贵公司办理业务，请给予接洽…… 云云。然后在信上敲上 A 公司的公章。

张三先生到了 B 公司后，把介绍信递给 B 公司的前台李四小姐。李小姐一看介绍信上有 A 公司的公章，而且 A 公司是经常和 B 公司有业务往来的，这位李小姐就相信张先生不是歹人了。

有的同学会问了：万一公章是伪造的，咋办捏？在此，要先声明，在本例子中，先假设

公章是难以伪造滴，否则故事没法说下去。

 引入中介机构的介绍信

好，回到刚才的话题。如果和 B 公司有业务往来的公司很多，每个公司的公章都不同，那前台就要懂得分辨各种公章，非常滴麻烦。所以，有某个中介公司 C，发现了这个商机。C 公司专门开设了一项“代理公章”的业务。

今后，A 公司的业务员去 B 公司，需要带 2 个介绍信：

介绍信 1

含有 C 公司的公章及 A 公司的公章。并且特地注明：C 公司信任 A 公司。


介绍信 2

仅含有 A 公司的公章，然后写上：兹有张三先生前往贵公司办理业务，请给予接洽……云云。

某些同学会问了，这样不是增加麻烦了吗？有啥好处捏？

主要的好处在于，对于接待公司的前台，就不需要记住各个公司的公章分别是啥样子的；他/她只要记住中介公司 C 的公章即可。当他/她拿到两份介绍信之后，先对“介绍信 1”的 C 公章，验明正身；确认无误之后，再比对“介绍信 1”和“介绍信 2”的两个 A 公章是否一致。如果是一样的，那就可以证明“介绍信 2”是可以信任的了。

## 2. 相关【专业术语】的解释

 “证书”是啥？

“证书”叫“digital certificate”或“public key certificate”。


它是用来证明某某东西确实是某某东西的东西（是不是像绕口令？）。通俗地说，证书就好比例子里面的公章。通过公章，可以证明该介绍信确实是对应的公司发出的。

理论上，人人都可以找个证书工具，自己做一个证书。那如何防止坏人自己制作证书出来骗人捏？请看后续 CA 的介绍。

 “CA”是啥？

CA 是“Certificate Authority”的缩写，也叫“证书授权中心”。

它是负责管理和签发证书的第三方机构，就好比例子里面的中介——C 公司。一般来说，CA 必须是所有行业 and 所有公众都信任的、认可的。因此它必须具有足够的权威性。就好比 A、B 两公司都必须信任 C 公司，才会找 C 公司作为公章的中介。

 “CA 证书”是啥？

CA 证书，顾名思义，就是 CA 颁发的证书。

前面已经说了，人人都可以找工具制作证书。但是你一个小破孩制作出来的证书是没啥用处的。因为你【不是】权威的 CA 机关，你自己搞的证书不具有权威性。

这就好比上述的例子里，某个坏人自己刻了一个公章，盖到介绍信上。但是别人一看，不是【受信任】的中介公司的公章，就不予理睬。坏蛋的阴谋就不能得逞啦。

 啥是证书之间的【信任关系】？

在前面的例子里谈到，引入中介后，业务员要同时带两个介绍信。第一个介绍信包含了两个公章，并注明，公章 C 信任公章 A。证书间的信任关系，就和这个类似。就是用—个证书来证明另一个证书是真实可信滴。

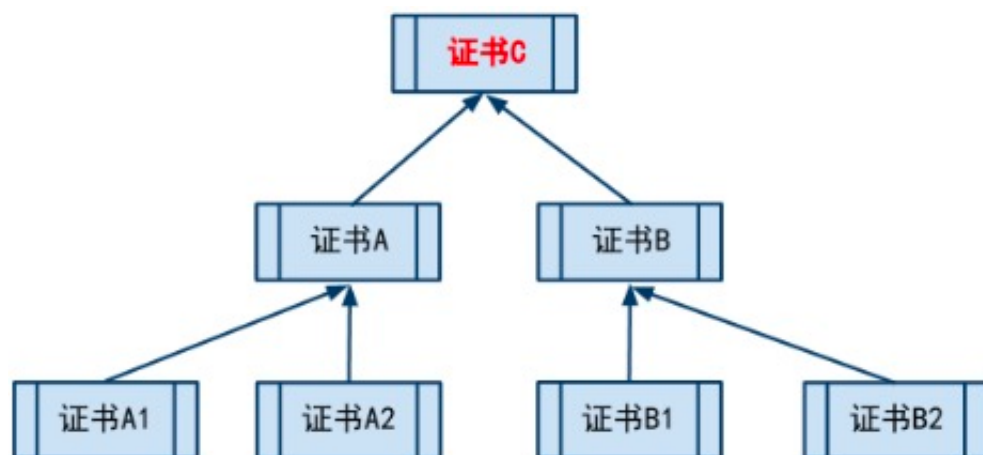
👤 啥是证书的【信任链】？

实际上，证书之间的信任关系，是可以嵌套的。比如，C 信任 A1，A1 信任 A2，A2 信任 A3..... 这个叫做证书的信任链。只要你信任链上的头一个证书，那后续的证书，都是可以信任滴。

👤 啥是【根证书】？

“根证书”的洋文叫“root certificate”。

假设 C 证书信任 A 和 B；然后 A 信任 A1 和 A2；B 信任 B1 和 B2。则它们之间，构成如下的一个树形关系（一个倒立的树）。



处于最顶上的树根位置的那个证书，就是“根证书”。除了根证书，其它证书都要依靠上一级的证书，来证明自己。那谁来证明“根证书”可靠捏？实际上，根证书自己证明自己是可靠滴（或者换句话说，根证书是不需要被证明滴）。

## • HTTPS 协议的【需求】是啥？

终于把背景知识说完了。下面正式进入正题。先来说说当初设计 HTTPS 是为了满足哪些需求？

👤 兼容性

因为是有 HTTP 再有 HTTPS。所以，HTTPS 的设计者肯定要考虑到对原有 HTTP 的兼容性。这里所说的兼容性包括很多方面。比如已有的 Web 应用要尽可能无缝地迁移到 HTTPS；比如对浏览器厂商而言，改动要尽可能小；.....

基于“兼容性”方面的考虑，很容易得出以下结论：

要单独使用一个新的协议，把 HTTP 协议包裹起来，（所谓的“HTTP over SSL”，实际上是在原有的 HTTP 数据外面加了一层 SSL 的封装。HTTP 协议原有的 GET、POST 之类的机制，基本上原封不动）

打个比方：如果原来的 HTTP 是塑料水管，容易被戳破；那么如今新设计的 HTTPS 就像是在原有的塑料水管之外，再包一层金属水管。一来，原有的塑料水管照样运行；二来，用金属加固了之后，不容易被戳破。

👤 可扩展性



前面说了，HTTPS 相当于是 “HTTP over SSL”。

如果 SSL 这个协议在 “可扩展性” 方面的设计足够牛逼，那么它除了能跟 HTTP 搭配，还能够跟其它的应用层协议搭配。岂不美哉？

现在看来，当初设计 SSL 的人确实比较牛。如今的 SSL/TLS 可以跟很多常用的应用层协议（比如：FTP、SMTP、POP、Telnet）搭配，以及我们今天讲的 DNS 协议，来强化这些应用层协议的安全性。

接着刚才打的比方：如果把 SSL/TLS 视作一根用来加固的金属管，它不仅可以用来加固输水的管道，还可以用来加固输煤气的管道。

### **保密性（防泄密）**

HTTPS 需要做到足够好的保密性。

说到保密性，首先要能够对抗 “嗅探”（Sniffer）。所谓的 “嗅探”，通俗而言就是监视你的网络传输流量。如果你使用【明文】的 HTTP 上网，那么监视者通过嗅探，就知道你在访问哪些网站的哪些页面。

嗅探是最低级的攻击手法。除了嗅探，HTTPS 还需要能对抗其它一些稍微高级的攻击手法 —— 比如 “重放攻击”。

### **完整性（防篡改）**

除了 “保密性”，还有一个同样重要的目标是 “确保完整性”。

在发明 HTTPS 之前，由于 HTTP 是明文的，不但容易被嗅探，还容易被篡改。

举个例子：

比如咱们的网络运营商（ISP）都比较流氓，以前通过 http 访问某网站（本来是没有广告的），竟然会跳出很多中国电信的广告。为啥会这样捏？因为你的网络流量需要经过 ISP 的线路才能到达公网。如果你使用的是明文的 HTTP，ISP 很容易就可以在你访问的页面中植入广告。所以，设计 HTTPS 的时候，还有一个需求是 “确保 HTTP 协议的内容【不】被篡改”。

### **真实性（防假冒）**

在谈到 HTTPS 的需求时，“真实性” 经常被忽略。其实 “真实性” 的重要程度【不亚于】前面的 “保密性” 和 “完整性”。

举个例子：

你因为使用网银，需要访问该网银的 Web 站点。那么，你如何确保你访问的网站确实是你想访问的网站？

有些同学会说：通过看网址里面的域名来确保。为啥说这样的同学是 “天真的”？因为 DNS 系统本身是不可靠的。由于 DNS 的不可靠（存在 “域名欺骗” 和 “域名劫持”），你看到的网址里面的域名【未必】是真实滴！

### **性能**

引入 HTTPS 之后，【不能】导致性能变得太差。否则的话，谁还愿意用？为了确保性能，SSL 的设计者至少要考虑如何选择加密算法（对称加密 or 非对称加密）？

## ● **设计 HTTPS 协议的主要【难点】是啥？**



设计 HTTPS 这个协议，有好几个难点。个人认为：“密钥交换”是最大的难点（没有之一）。在传统的密码学场景中，假如张三要跟李四建立一个加密通讯的渠道，双方事先要约定好使用哪种加密算法？同时也要约定好使用的密钥是啥？在这个场景中，加密算法的【类型】让旁人知道，没太大关系。但是密钥【千万不能】让旁人知道。一旦旁人知道了密钥，自然就可以破解通讯的密文，得到明文。

回到 HTTPS 的场景。

当你访问某个公网的网站，你的浏览器和网站的服务器之间，如果要建立加密通讯，必然要商量好双方使用啥算法，啥密钥。——在网络通讯术语中，这个过程称之为“握手”（叫“handshake”）。在握手阶段，因为加密方式还没有协商好，所以握手阶段的通讯必定是【明文】滴！既然是明文，自然有可能被第三方偷窥到。然后，还要考虑到双方之间隔着一个【互联网】，啥样的事情都可能发生（不光会有“数据偷窥”，还会有【数据篡改】）。因此，在握手的过程中，如何做到安全地交换密钥信息，而不让周围的第三方看到。这就是设计 HTTPS 最大的难点。

### 1. 方案 1——单纯用“对称加密算法”的可行性

首先简单阐述一下，“单纯用对称加密”为啥是【不可行】滴。

如果“单纯用对称加密”，浏览器和网站之间势必先要交换“对称加密的密钥”。如果这个密钥直接用【明文】传输，很容易就会被第三方（有可能是“攻击者”）偷窥到；如果这个密钥用密文传输，那就再次引入了“如何交换加密密钥”的问题——这就变成“先有鸡还是先有蛋”的循环逻辑了。所以，【单纯用】对称加密，是没戏滴。

### 2. 方案 2——单纯用“非对称加密算法”的风险

说完“对称加密”，再来说说“非对称加密”。

#### 第 1 步

网站服务器先基于“【非】对称加密算法”，随机生成一个“密钥对”（为叙述方便，称之为“k1 和 k2”）。因为是随机生成的，目前为止，只有网站服务器才知道 k1 和 k2。

#### 第 2 步

网站把 k1 保留在自己手中，把 k2 用【明文】的方式发送给访问者的浏览器。

因为 k2 是明文发送的，自然有可能被偷窥。不过不要紧。即使偷窥者拿到 k2，也【极难】根据 k2 推算出 k1（注：这是由“非对称加密算法”从数学上保证滴）

#### 第 3 步

浏览器拿到 k2 之后，先【随机生成】第三个对称加密的密钥（简称 k）。

然后用 k2 加密 k，得到 k'（k' 是 k 的加密结果）浏览器把 k' 发送给网站服务器。

由于 k1 和 k2 是成对的，所以只有 k1 才能解密 k2 的加密结果。

因此这个过程中，即使被第三方偷窥，第三方也【无法】从 k' 解密得到 k

#### 第 4 步

网站服务器拿到 k' 之后，用 k1 进行解密，得到 k

至此，浏览器和网站服务器就完成了密钥交换，双方都知道 k，而且【貌似】第三方无法拿到 k

然后，双方就可以用  $k$  来进行数据双向传输的加密。

现在，留一点【思考时间】——大家觉得上述过程是否严密？如果不严密，漏洞在哪里？

**“方案 2” 依然是【不】安全滴**——虽然“方案 2”可以在一定程度上防止网络数据的“偷窥 / 嗅探”，但是【无法】防范网络数据的【篡改】。

假设有一个攻击者处于“浏览器”和“网站服务器”的通讯线路之间，并且这个攻击者具备“【修改】双方传输数据”的能力。那么，这个攻击者就可以攻破“方案 2”。具体的攻击过程如下：

### 第 1 步

这一步跟原先一样——服务器先随机生成一个“非对称的密钥对”  $k_1$  和  $k_2$ （此时只有网站知道  $k_1$  和  $k_2$ ）

### 第 2 步

当网站发送  $k_2$  给浏览器的时候，攻击者截获  $k_2$ ，保留在自己手上。

然后攻击者自己生成一个【伪造的】密钥对（以下称为  $pk_1$  和  $pk_2$ ）。

攻击者把  $pk_2$  发送给浏览器。

### 第 3 步

浏览器收到  $pk_2$ ，以为  $pk_2$  就是网站发送的。

浏览器不知情，依旧随机生成一个对称加密的密钥  $k$ ，然后用  $pk_2$  加密  $k$ ，得到密文的  $k'$ 。浏览器把  $k'$  发送给网站。

（以下是关键）

发送的过程中，再次被攻击者截获。

因为  $pk_1$   $pk_2$  都是攻击者自己生成的，所以攻击者自然就可以用  $pk_1$  来解密  $k'$  得到  $k$ 。然后，攻击者拿到  $k$  之后，用之前截获的  $k_2$  重新加密，得到  $k''$ ，并把  $k''$  发送给网站。


### 第 4 步

网站服务器收到了  $k''$  之后，用自己保存的  $k_1$  可以正常解密，所以网站方面不会起疑心。

至此，攻击者完成了一次漂亮的偷梁换柱，而且让双方都没有起疑心。

**上述过程，也就是传说中大名鼎鼎的【中间人攻击】**（“Man-In-The-Middle attack”，缩写是 MITM）。

“中间人攻击”有很多种“类型”，刚才演示的是针对“【单纯的】非对称加密”的中间人攻击。

 方案 2 失败的根源——缺乏【可靠的】身份认证

为啥“方案 2”会失败捏？

除了俺在图中提到的“攻击者具备篡改数据的能力”，还有另一点关键点——“方案 2 缺乏身份认证机制”。

正是因为“缺乏身份认证机制”，所以当攻击者一开始截获  $k_2$  并把自己伪造的  $pk_2$  发送给浏览器时，浏览器无法鉴别：自己收到的密钥是不是真的来自于网站服务器。

假如具备某种【可靠的】身份认证机制，即使攻击者能够篡改数据，但是篡改之后的数

据很容易被识破。那篡改也就失去了意义。

### 【身份认证】的几种方式

下面，介绍几种常见的“身份认证原理”。

#### ■ 基于某些“私密的共享信息”

为了解释“私密的共享信息”这个概念，咱们先抛开“信息安全”，谈谈日常生活中的某个场景。

假设你有一个久未联系的老朋友。因为时间久远，你已经没有此人的联系方式了。某天，此人突然给你发了一封电子邮件。

那么，你如何确保——发邮件的人确实是你的老朋友捏？

有一个办法就是：你用邮件向对方询问某个私密的事情（这个事情只有你和你的这个朋友知道，其他人不知道）。如果对方能够回答出来，那么对方

【很有可能】确实是你的老朋友。

从这个例子可以看出，如果通讯双方具有某些“私密的共享信息”（只有双方知道，第三方不知道），就能以此为基础，进行身份认证，从而建立信任。

#### ■ 基于双方都信任的“公证人”

“私密的共享信息”，通常需要双方互相比较熟悉，才行得通。如果双方本来就互不相识，如何进行身份认证以建立信任关系捏？

这时候还有另一个办法——依靠双方都信任的某个“公证人”来建立信任关系。

如今 C2C 模式的电子商务，其实用的就是这种方式——由电商平台充当公证人，让买家与卖家建立某种程度的信任关系。

### 如何解决 SSL 的【身份认证】问题——CA 的引入

说完身份认证的方式/原理，再回到 SSL/TLS 的话题上。

对于 SSL/TLS 的应用场景，由于双方（“浏览器”和“网站服务器”）通常都是素不相识滴，显然【不可能】采用第一种方式（私密的共享信息），而只能采用第二种方式

（依赖双方都信任的“公证人”）。那么，谁来充当这个公证人捏？这时候，CA 就华丽地登场啦。

## 3. 方案 3——基于 CA 证书进行密钥交换

其实“方案 3”跟“方案 2”很像的，主要差别在于——“方案 3”增加了“CA 数字证书”这个环节。所谓的数字证书，技术上依赖的还是前面提到的“非对称加密”。为了描述“CA 证书”在 SSL/TLS 中的作用，俺大致说一下原理（仅仅是原理，具体的技术实现要略复杂些）：

### 第 1 步（这是“一次性”的准备工作）

网站方面首先要花一笔银子，在某个 CA 那里购买一个数字证书。

该证书通常会对应几个文件：其中一个文件包含公钥，还有一个文件包含私钥。

此处的“私钥”，相当于“方案 2”里面的  $k_1$ ；而“公钥”类似于“方案 2”里面的

k2。

网站方面必须在 Web 服务器上部署这两个文件。

所谓的“公钥”，顾名思义就是可以公开的 key；而所谓的“私钥”就是私密的 key。

“非对称加密算法”从数学上确保了——即使你知道某个公钥，也很难（不是不可能，是很难）根据此公钥推导出对应的私钥。

## 第 2 步

当浏览器访问该网站，Web 服务器首先把包含公钥的证书发送给浏览器。

## 第 3 步

浏览器验证网站发过来的证书。如果发现其中有诈，浏览器会提示“CA 证书安全警告”。

由于有了这一步，就大大降低了（注意：是“大大降低”，而不是“彻底消除”）前面提到的“中间人攻击”的风险。

## 为啥浏览器能发现 CA 证书是否有诈？

因为正经的 CA 证书，都是来自某个权威的 CA。如果某个 CA 足够权威，那么主流的操作系统（或浏览器）会内置该 CA 的“根证书”。

（比如 Windows 中就内置了几十个权威 CA 的根证书）

因此，浏览器就可以利用系统内置的根证书，来判断网站发过来的 CA 证书是不是某个 CA 颁发的。

## 第 4 步

如果网站发过来的 CA 证书没有问题，那么浏览器就从该 CA 证书中提取出“公钥”。然后浏览器随机生成一个“对称加密的密钥”（以下称为 k）。用 CA 证书的公钥加密 k，得到密文 k'。

浏览器把 k' 发送给网站。

## 第 5 步

网站收到浏览器发过来的 k'，用服务器上的私钥进行解密，得到 k。

至此，浏览器和网站都拥有 k，“密钥交换”大功告成啦。

# DNS over HTTPS

“DNS over HTTPS”有时也被简称为【DoH】。本文以下部分用 DoH 来称呼之。

## 协议栈

```
1  -----
2      DoH
3  -----
4      HTTP
5  -----
6      TLS
7  -----
8      TCP
9  -----
10     IP
11  -----
```

## 安全性原理

顾名思义，DNS over HTTPS 就是基于 HTTPS 隧道之上的域名协议。而 HTTPS 又是 “HTTP over TLS”。所以 DoH 相当于是【双重隧道】的协议。

与 DoT 类似，DoH 最终也是依靠 TLS 来实现了【保密性】与【完整性】。

## 自建 DNS 服务器

1. 全链路 TLS
2. 增加缓存
3. 配置信任的上游 DNS 服务器
4. 可定制性高