一文读懂HTTPS ⭐揭秘加密传输背后的原理与Nginx配置攻略

HTTP协议使用明文传输，不能保证安全性，而使用HTTPS却能够保证传输安全

为什么HTTPS能够保证传输安全呢?

HTTPS在HTTP的基础上除了要进行TCP三次握手，还会进行TLS的四次握手

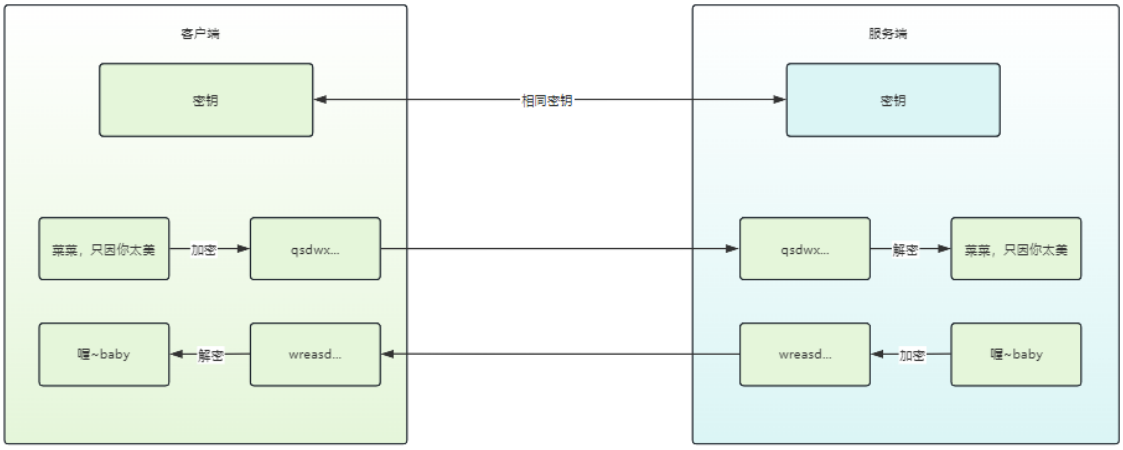
（本文不对HTTP进行说明）

在进行TLS四次握手时会**使用对称加密、非对称加密、数字证书的验证**等多种方式一起来保证连接的安全

对称加密

在对称加密模型中，加密和解密使用的是同一个密钥

客户端使用密钥将明文转化为密文，服务端再使用相同的密钥将密文转化为明文



对称加密非常**高效适合加密大量数据**，但如果第三方恶意机构也拥有密钥就会变得不安全

非对称加密

在非对称加密模型中分为公钥和私钥，**公钥（可以暴露给外界），私钥（自己留着）**

客户端先向服务端申请公钥，客户端使用公钥将明文转换为密文，服务端再使用私钥将密文转换为明文

非对称加密加密效率不如对称加密，开销会更大

但非对称加密相对安全，即使公钥暴露，也不能拦截客户端请求进行解密

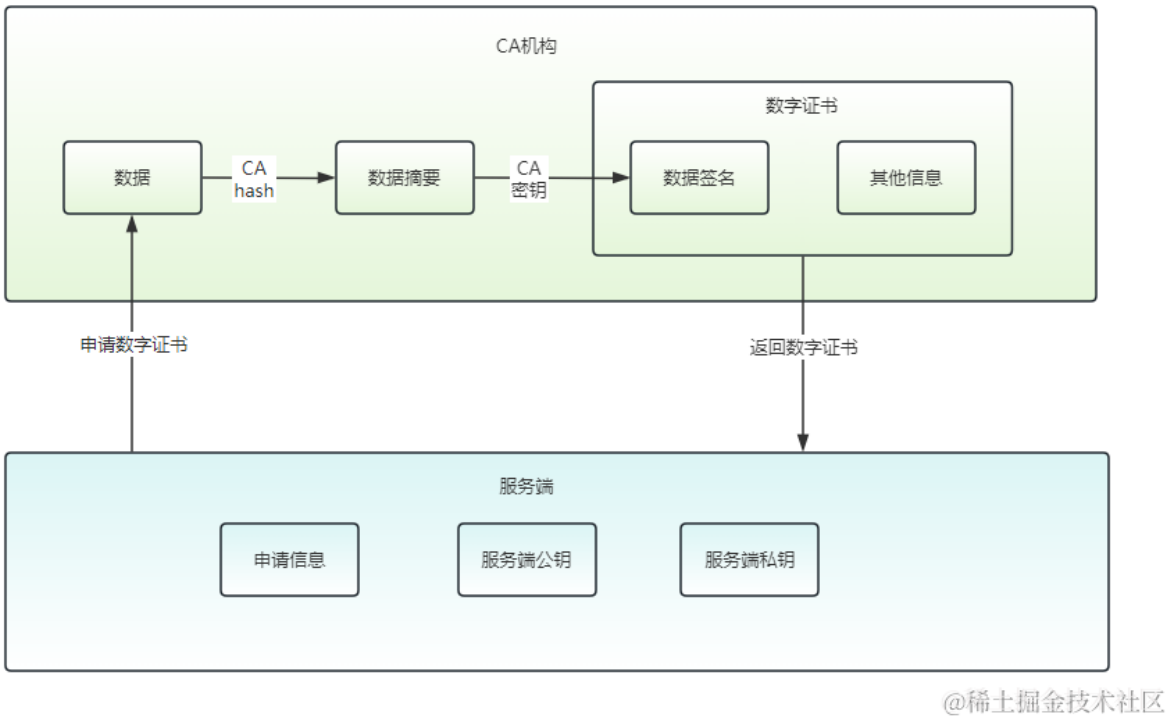
但第三方恶意机构可以伪装成服务端，对数据进行篡改并发送给真正的服务端，同时向客户端发访伪造公钥

为了解决这种问题，客户端、服务端要信任第三方权威机构并申请证书

申请证书

申请证书指的是服务端向第三方权威机构（CA）申请数字证书，有了数字证书后，客户端能够判断**数据是否被篡改和公钥是否可信任**

1. 服务端生成公钥、私钥
2. 服务端向CA申请证书
3. CA机构**先使用hash得到数据摘要，再用自己的私钥加密生成数字签名，配合服务端公钥、申请信息等其他信息生成数字证书**再返回



验证证书

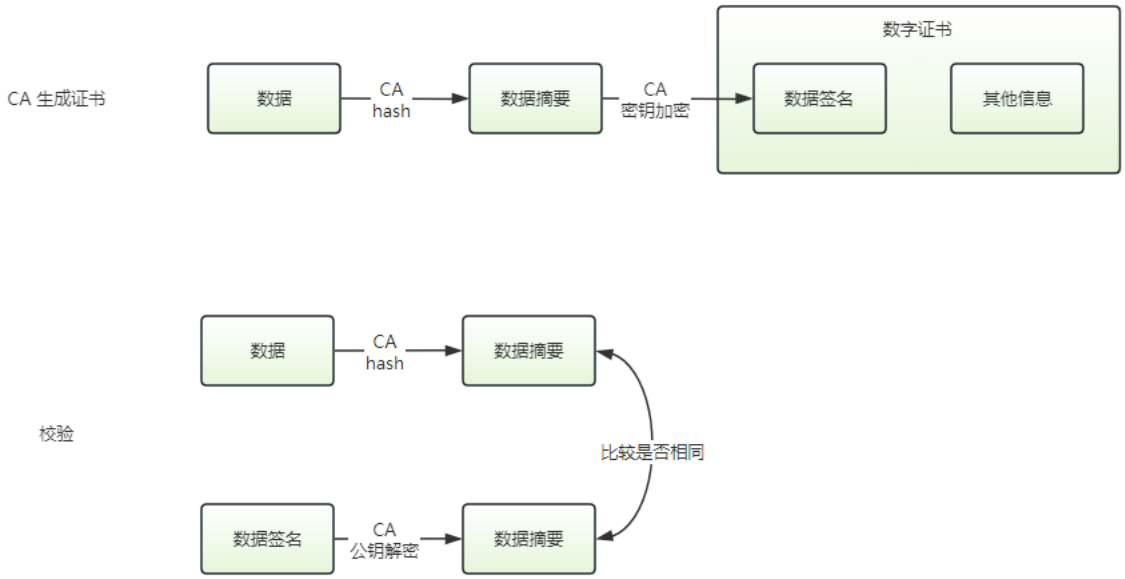
客户端收到证书后，该如何验证数据是否被篡改？公钥是否可信呢？

由于证书中数字签名是由CA私钥加密的，那么解密就需要CA公钥

客户端首先要信任CA机构，必须有CA机构的根证书（主流OS或浏览器已内置）

客户端拿到服务端的证书后，**使用CA公钥解密数字签名得到数据摘要，再使用生成证书时相同的hash加密数据获得数据摘要，再进行比较**

（CA 公钥和hash 都是来自根证书）



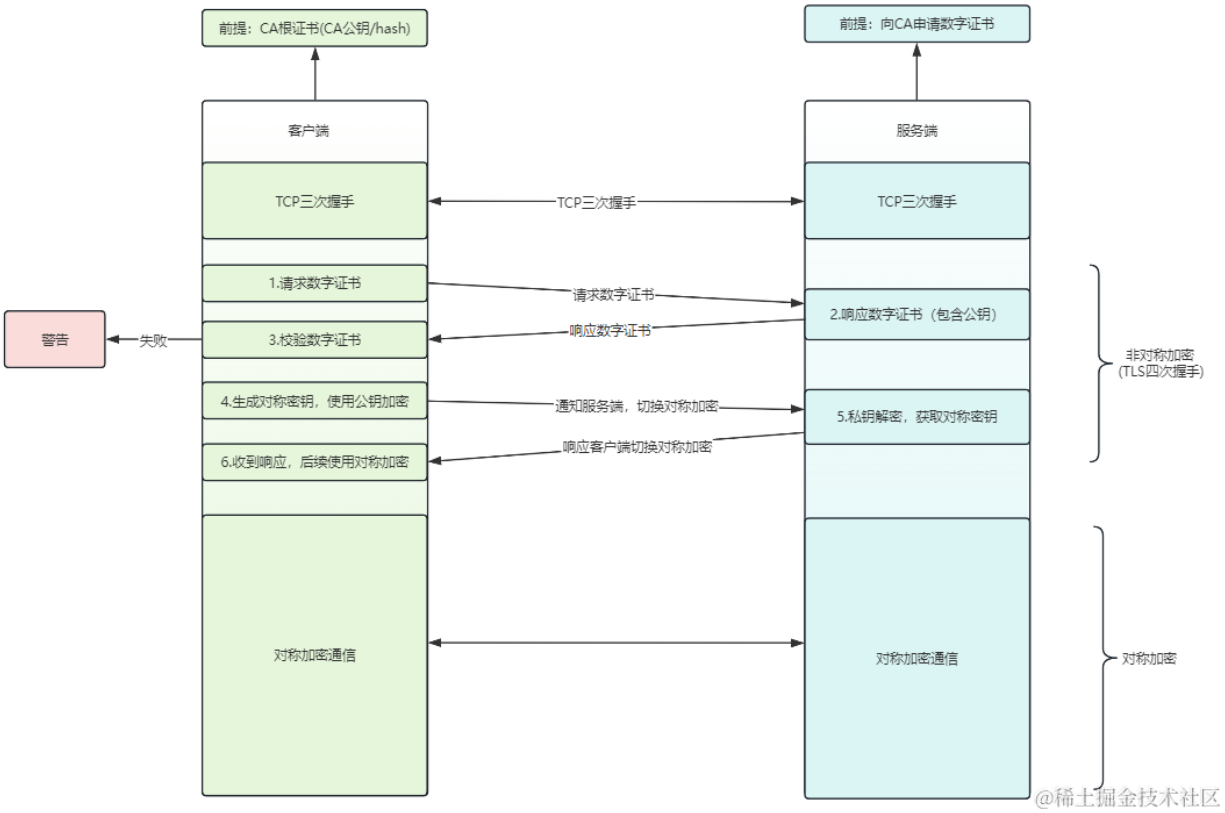
如果不相等说明数据被篡改，让浏览器提示警告：证书不可信任

如果数据被篡改、公钥被替换，它们生成的数据摘要都会不相等，从而发出警告

HTTPS流程

HTTPS建立连接的流程会进行TCP三次握手+TSL四次握手

1. TCP三次握手
2. 客户端请求服务端证书
3. 服务端返回证书
4. 客户端将证书中的数字签名使用CA公钥解密得到数据摘要，再使用CA相同的hash加密信息得到数据摘要，两个数据摘要相同说明数据未被篡改/公钥可信任
5. 验证成功，根据服务端公钥生成对称密钥发送服务端
6. 服务端使用私钥解密获取对称密钥，后续使用对称密钥加密/解密



在HTTPS中**先请求证书、再校验证书、最后生成对称密钥**，基于对称密钥高效的特点，即使后续通信数据量大，性能也不会太差

请求证书和校验证书可以看成非对称加密，因此**HTTPS是通过第三方可信任机构、数字证书、非对称加密、对称加密一起实现的**

nginx配置HTTPS

如果要使用HTTPS，可以在nginx中进行相关配置

其中 **ssl_certificate 为数字证书的路径、ssl_certificate_key为服务端私钥的路径**（这两个文件可以由云服务器或OpenSSL生成）

```
# ----------HTTPS配置-----------
server {
    # 监听HTTPS默认的443端口
    listen 443;
    # 配置自己项目的域名
    server_name www.xxx.com;
    # 打开SSL加密传输
    ssl on;
    # 输入域名后，首页文件所在的目录
    root html;
    # 配置首页的文件名
    index index.html index.htm index.jsp index.ftl;
    # 配置 数字证书
    ssl_certificate   certificate/xxx.pem;
```

```
    # 配置 服务器私钥
    ssl_certificate_key certificate/xxx.key;
    # 停止通信时，加密会话的有效期，在该时间段内不需要重新交换密钥
    ssl_session_timeout 5m;
    # TLS握手时，服务器采用的密码套件
    ssl_ciphers ECDHE-RSA-AES128-GCM-
SHA256:ECDHE:ECDH:AES:HIGH:!NULL:!aNULL:!MD5:!ADH:!RC4;
    # 服务器支持的TLS版本
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    # 开启由服务器决定采用的密码套件
    ssl_prefer_server_ciphers on;


    location / {
        try_files $uri $uri/ /html/index.html;
    }

}

# ---------HTTP请求转HTTPS-------------
server {
    # 监听HTTP默认的80端口
    listen 80;
    # 如果80端口出现访问该域名的请求
    server_name www.xxx.com;
    # 将请求改写为HTTPS（这里写你配置了HTTPS的域名）
    rewrite ^(.*)$ https://www.xxx.com;
}
```

总结

对称加密模型只有一种密钥，加密/解密都使用这种密钥，高效但不安全

非对称加密模型分为公钥与私钥，公钥可以暴露给外界，而私钥自己保存，加密/解密开销大，但只是相对安全，遇到恶意第三方结构伪造成服务端篡改数据/伪造公钥还是不安全的

为了让客户端判断数据是否被篡改/公钥是否可信任，引入第三方权威可信机构

申请证书时，CA根据hash对数据进行加密得到数据摘要，再使用密钥对数据摘要进行加密得到数字签名，数字签名与其他信息形成数字证书

验证数字证书时，根据CA hash加密数据得到的数据摘要和使用CA公钥对证书中签名解密得到的数据摘要进行对比，不相同说明数据被篡改或公钥不可信，从而发出警告

客户端、服务端请求前都依赖第三方可信机构，客户端依赖CA根证书（CA hash、公钥），服务端依赖数据证书

HTTPS通过可信任的第三方结构、数字证书、非对称加密、对称加密等多种方式实现

最后（不要白嫖，一键三连求求拉~）

本篇文章被收入专栏 `Effective Java`，感兴趣的同学可以持续关注喔

本篇文章笔记以及案例被收入 `Gitee-CaiCaiJava`、 `Github-CaiCaiJava`，除此之外还有更多Java进阶相关知识，感兴趣的同学可以starred持续关注喔~

有什么问题可以在评论区交流，如果觉得菜菜写的不错，可以点赞、关注、收藏支持一下~

关注菜菜，分享更多技术干货，公众号：菜菜的后端私房菜