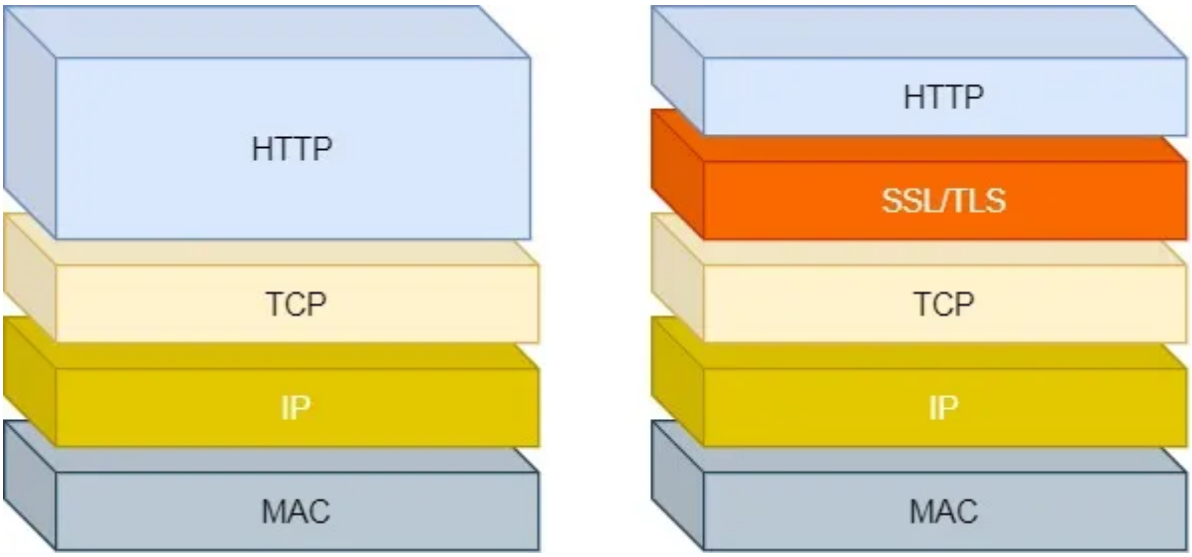
应用层 - HTTPS

HTTPS 概述

HTTPS概述

- 采用**混合加密**——达成**信息加密**——规避**窃听风险**——实现**机密性**，
- 采用**摘要算法**——达成**校验机制**——规避**篡改风险**——实现**报文完整性**，
- 采用**数字证书**——达成**身份证书**——规避**冒充风险**——实现**端点鉴别**，

HTTP**S** 在 HTTP 与 TCP 层之间加入了 `SSL/TLS` 协议。



HTTP 与 HTTPS

HTTP与HTTPS的区别

- 安全性
  - HTTP 是超文本传输协议，信息是明文传输，存在安全风险的问题。
  - HTTPS 则解决 HTTP 不安全的缺陷，在 TCP 和 HTTP 网络层之间加入了 SSL/TLS 安全协议，使得报文能够加密传输。
- 连接建立过程
  - HTTP 连接建立相对简单， TCP 三次握手之后便可进行 HTTP 的报文传输。
  - 而 HTTPS 在 TCP 三次握手之后，还需进行 SSL/TLS 的握手过程，才可进入加密报文传输。
- 端口号
  - HTTP 的端口号是 80
  - HTTPS 的端口号是 443。
- HTTPS 协议需要向 CA（证书权威机构）申请数字证书，来保证服务器的身份是可信的。

HTTPS 的缺点

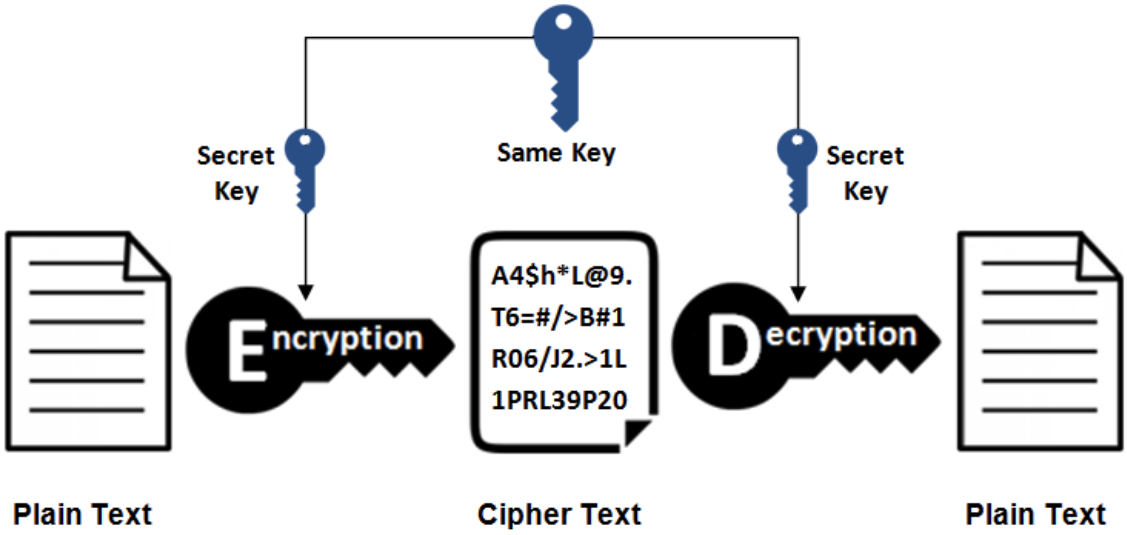
- 因为需要进行加密解密等过程，因此速度会更慢；
- 需要支付证书授权的高额费用。

加密算法

对称密钥加密

对称密钥加密（Symmetric-Key Encryption），加密和解密使用同一密钥。

- 优点：运算速度快；
- 缺点：无法安全地将密钥传输给通信方。



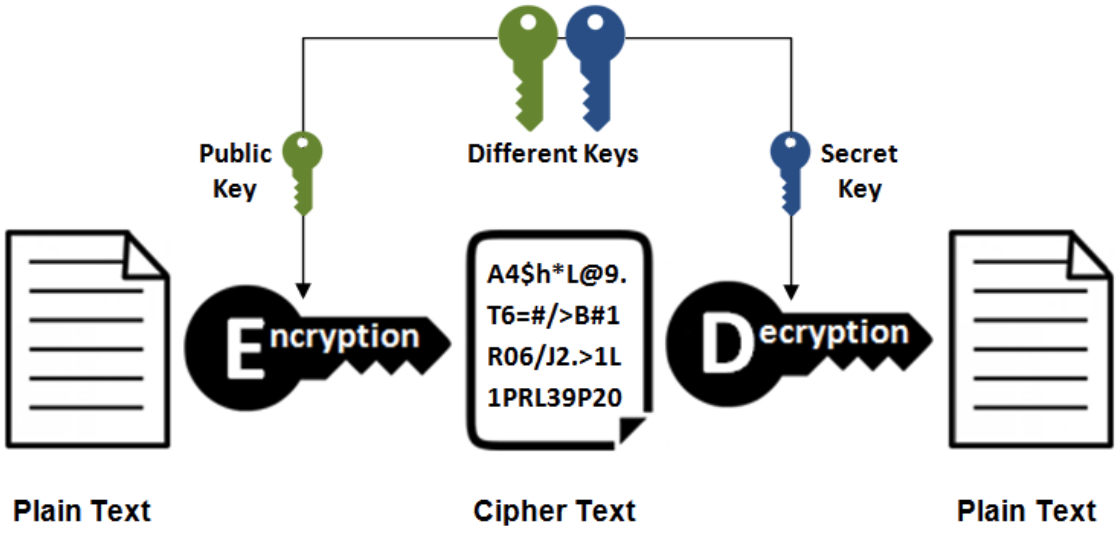
非对称密钥加密

非对称密钥加密，又称公开密钥加密（Public-Key Encryption），加密和解密使用不同的密钥。

公开密钥所有人都可以获得，通信发送方获得接收方的公开密钥之后，就可以使用公开密钥进行加密，接收方收到通信内容后使用私有密钥解密。

非对称密钥除了用来加密，还可以用来进行签名。因为私有密钥无法被其他人获取，因此通信发送方使用其私有密钥进行签名，通信接收方使用发送方的公开密钥对签名进行解密，就能判断这个签名是否正确。

- 优点：可以更安全地将公开密钥传输给通信发送方；
- 缺点：运算速度慢。



混合加密

HTTPS通过**混合加密**的方式可以保证信息的**机密性**，解决了窃听的风险。

混合加密

HTTPS 采用的是**对称加密**和**非对称加密**结合的「混合加密」方式：

- 在通信建立前采用**非对称加密**的方式交换「会话秘钥」，后续就不再使用非对称加密。
- 在通信过程中全部使用**对称加密**的「会话秘钥」的方式加密明文数据。

采用「混合加密」的方式的原因：

- **对称加密**只使用一个密钥，运算速度快，密钥必须保密，无法做到安全的密钥交换。
- **非对称加密**使用两个密钥：公钥和私钥，公钥可以任意分发而私钥保密，解决了密钥交换问题但速度慢。

摘要算法

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」，用于校验数据的完整性，解决了篡改的风险。

校验完整性

客户端在发送明文之前会通过摘要算法算出明文的「指纹」，发送的时候把「指纹 + 明文」一同加密成密文后，发送给服务器，服务器解密后，用相同的摘要算法算出发送过来的明文，通过比较客户端携带的「指纹」和当前算出的「指纹」做比较，若「指纹」相同，说明数据是完整的。

数字证书

数字证书和 CA 机构

一个数字证书通常包含了：

- 公钥；
- 持有者信息；
- 证书认证机构（CA）的信息；
- CA 对这份文件的数字签名及使用的算法；
- 证书有效期；
- 还有一些其他额外信息；

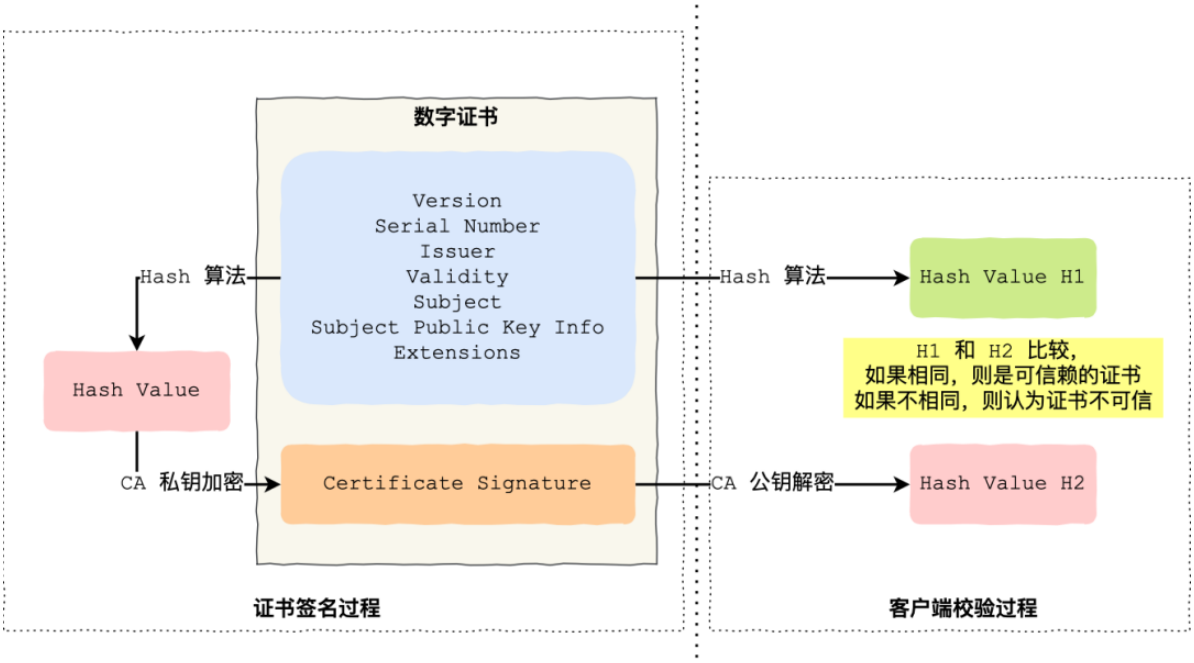
那数字证书的作用，是用来认证公钥持有者的身份，以防止第三方进行冒充。说简单些，证书就是用来告诉客户端，该服务端是否是合法的，因为只有证书合法，才代表服务端身份是可信的。

> 我们用证书来认证公钥持有者的身份（服务端的身份），那证书又是怎么来的？

为了让服务端的公钥被大家信任，服务端的证书都是由 **CA （*Certificate Authority*，证书认证机构）**签名的，CA 就是网络世界里的公安局、公证中心，具有极高的可信度，所以由它来给各个公钥签名，信任的一方签发的证书，那必然证书也是被信任的。之所以要签名，是因为签名的作用可以避免中间人在获取证书时对证书内容的篡改。

数字证书签发和验证流程

如下图图所示，为数字证书签发和验证流程：



CA **签发证书**的过程，如上图左边部分：

- 首先 CA 会把持有者的公钥、用途、颁发者、有效时间等信息打成一个包，然后对这些信息进行 Hash 计算，得到一个 Hash 值；
- 然后 CA 会使用自己的私钥将该 Hash 值加密，生成 Certificate Signature，也就是 CA 对证书做了签名；
- 最后将 Certificate Signature 添加在文件证书上，形成数字证书；

客户端**校验证书**的过程，如上图右边部分：

- 首先客户端会使用同样的 Hash 算法获取该证书的 Hash 值 H1；
- 通常浏览器和操作系统中集成了 CA 的公钥信息，浏览器收到证书后可以使用 CA 的公钥解密 Certificate Signature 内容，得到一个 Hash 值 H2 ；
- 最后比较 H1 和 H2，如果值相同，则为可信赖的证书，否则则认为证书不可信。

证书链

但事实上，证书的验证过程中还存在一个证书信任链的问题，因为我们向 CA 申请的证书一般不是根证书签发的，而是由中间证书签发的，比如百度的证书，从下图你可以看到，证书的层级有三级：

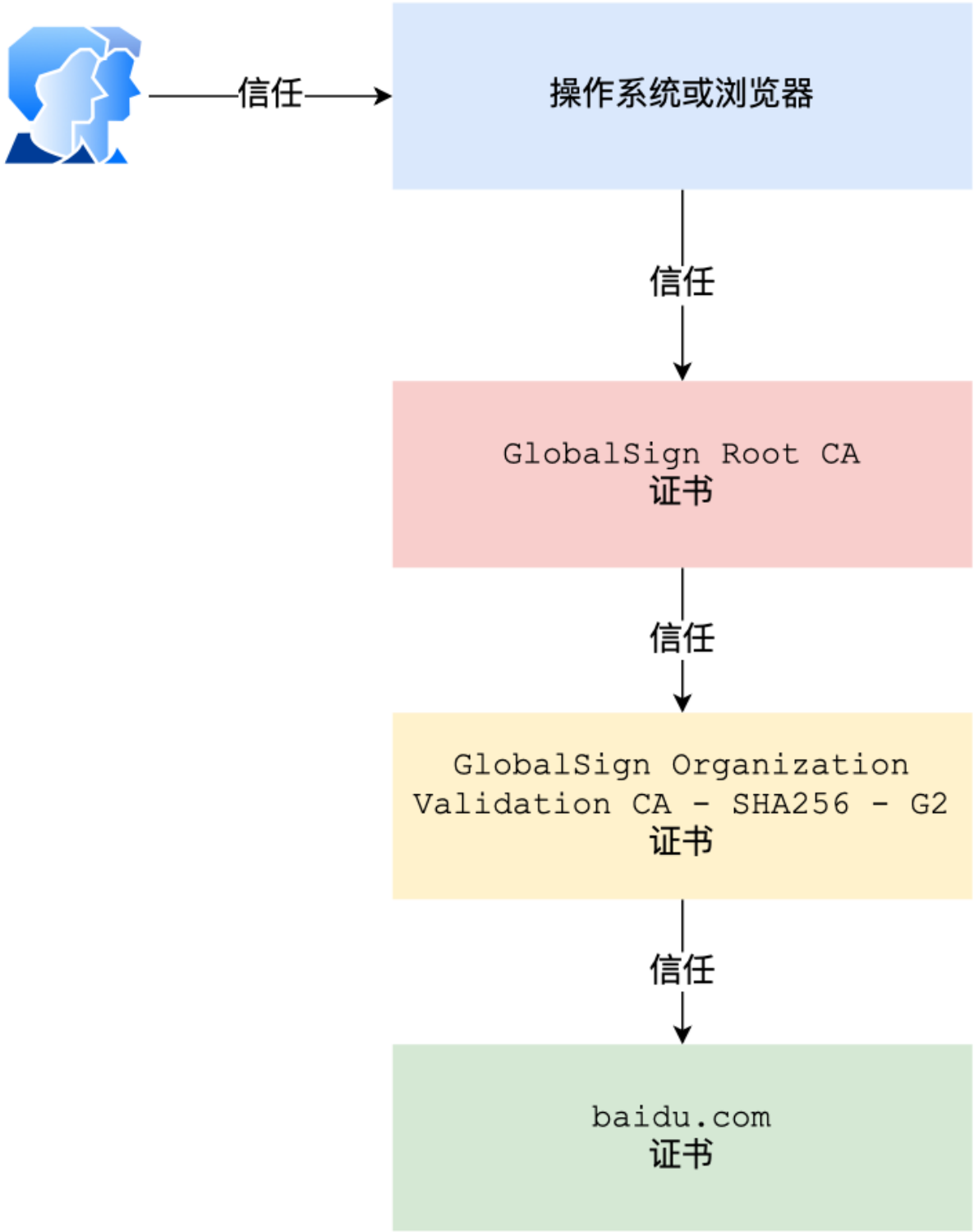
对于这种三级层级关系的证书的验证过程如下：

- 客户端收到 baidu.com 的证书后，发现这个证书的签发者不是根证书，就无法根据本地已有的根证书中的公钥去验证 baidu.com 证书是否可信。于是，客户端根据 baidu.com 证书中的签发者，找到该证书的颁发机构是 “GlobalSign Organization Validation CA - SHA256 - G2”，然后向 CA 请求该中间证书。

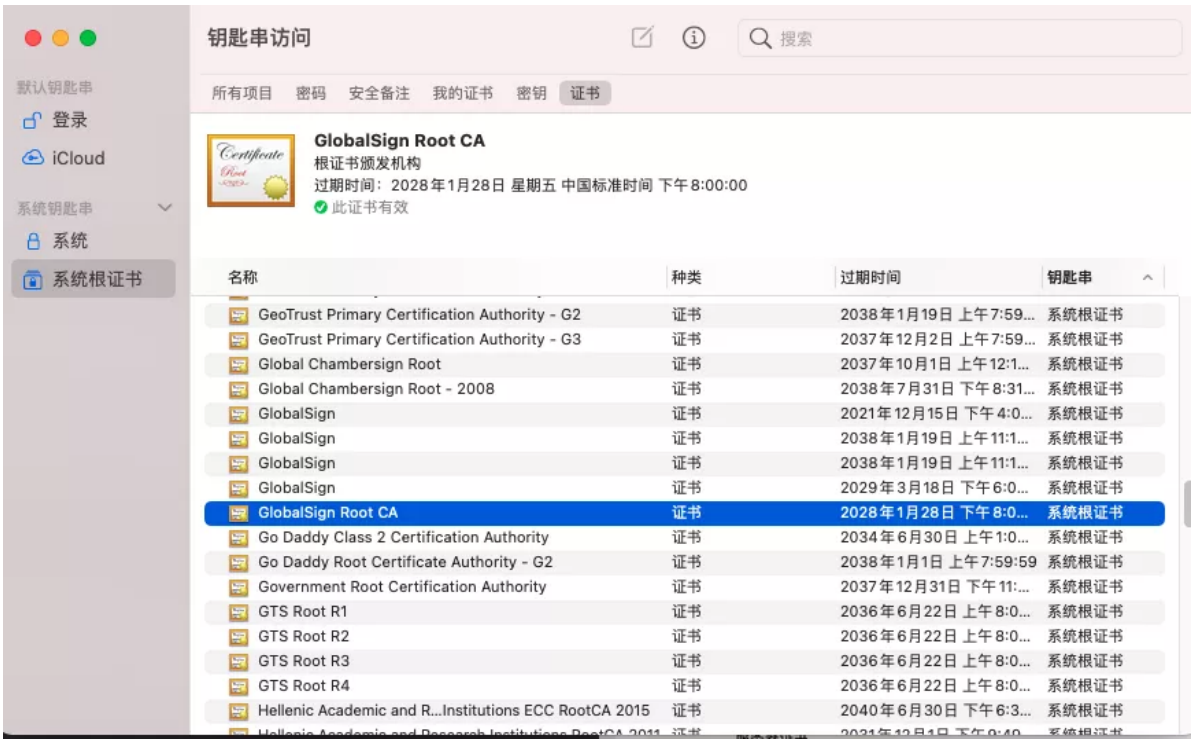
- 请求到证书后发现 “GlobalSign Organization Validation CA - SHA256 - G2” 证书是由 “GlobalSign Root CA” 签发的，由于 “GlobalSign Root CA” 没有再上级签发机构，说明它是根证书，也就是自签证书。应用软件会检查此证书有否已预载于根证书清单上，如果有，则可以利用根证书中的公钥去验证 “GlobalSign Organization Validation CA - SHA256 - G2” 证书，如果发现验证通过，就认为该中间证书是可信的。
- “GlobalSign Organization Validation CA - SHA256 - G2” 证书被信任后，可以使用 “GlobalSign Organization Validation CA - SHA256 - G2” 证书中的公钥去验证 baidu.com 证书的可信性，如果验证通过，就可以信任 baidu.com 证书。

在这四个步骤中，最开始客户端只信任根证书 GlobalSign Root CA 证书的，然后 “GlobalSign Root CA” 证书信任 “GlobalSign Organization Validation CA - SHA256 - G2” 证书，而 “GlobalSign Organization Validation CA - SHA256 - G2” 证书又信任 baidu.com 证书，于是客户端也信任 baidu.com 证书。

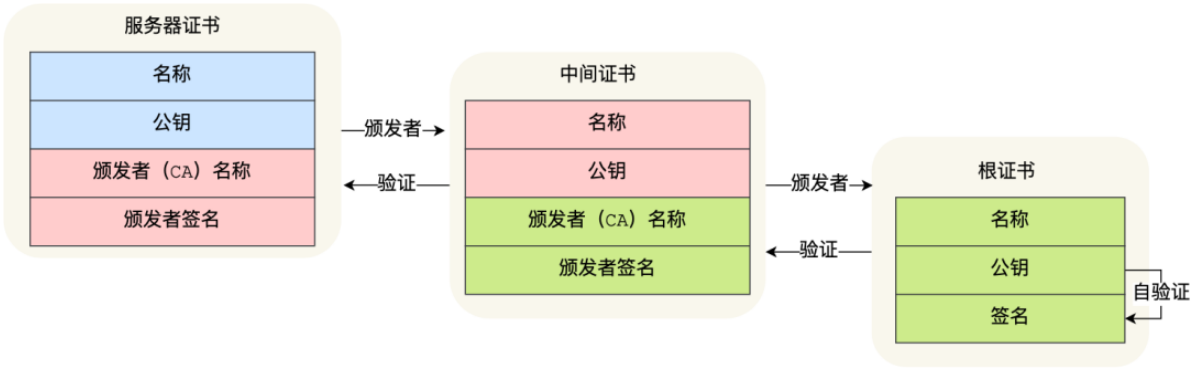
总括来说，由于用户信任 GlobalSign，所以由 GlobalSign 所担保的 baidu.com 可以被信任，另外由于用户信任操作系统或浏览器的软件商，所以由软件商预载了根证书的 GlobalSign 都可被信任。



操作系统里一般都会内置一些根证书，比如我的 MAC 电脑里内置的根证书有这么多：



这样的一层层地验证就构成了一条信任链路，整个证书信任链验证流程如下图所示：



> 为什么需要证书链这么麻烦的流程？
>
> Root CA 为什么不直接颁发证书，而是要搞那么多中间层级呢？

这是为了确保根证书的绝对安全性，将根证书隔离地越严格越好，不然根证书如果失守了，那么整个信任链都会有问题。

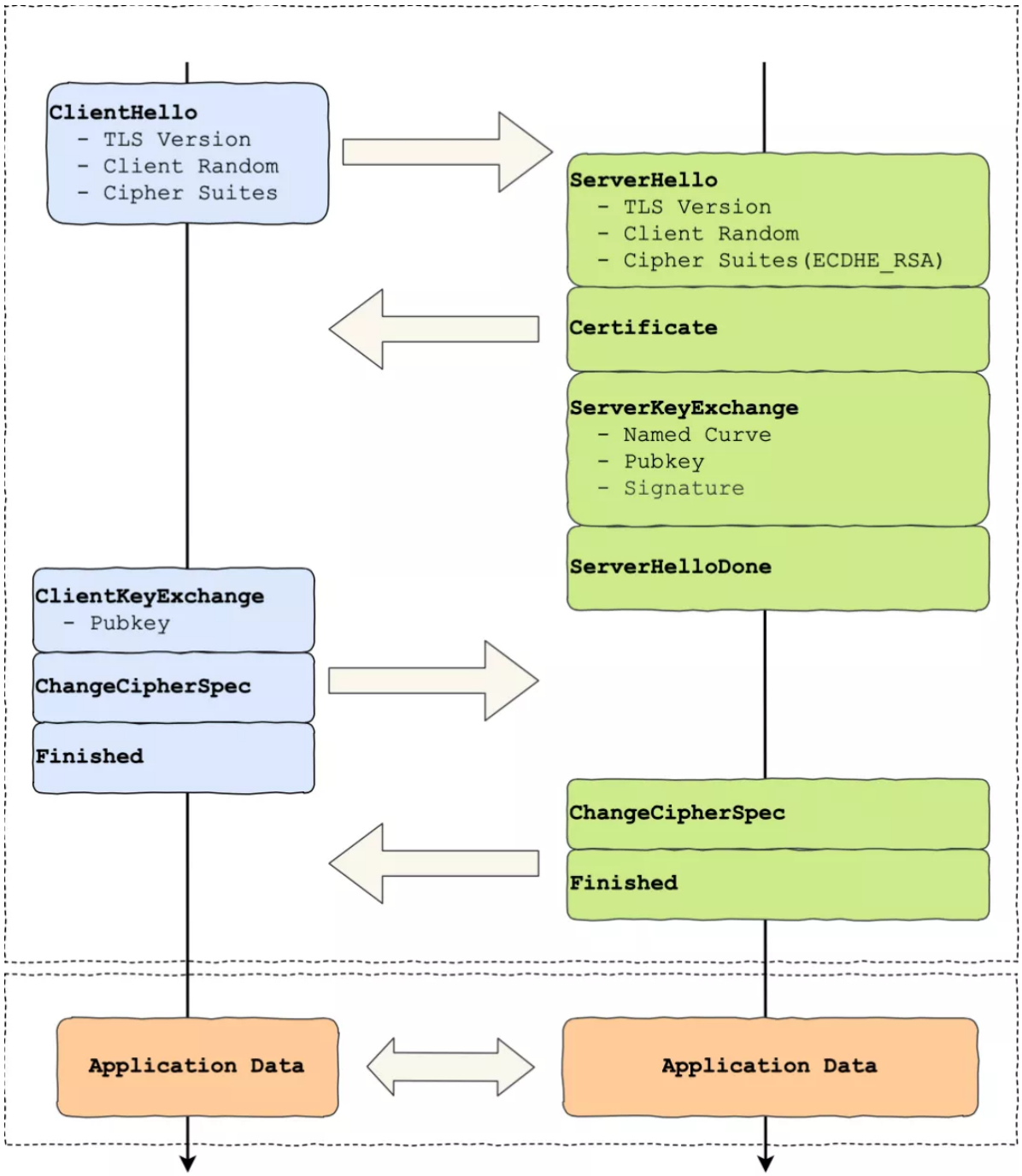
TLS握手过程

握手概述

SSL/TLS 协议基本流程：

- 客户端向服务器索要并验证服务器的证书。
- 双方协商生产「会话秘钥」。
- 双方采用「会话秘钥」进行加密通信。

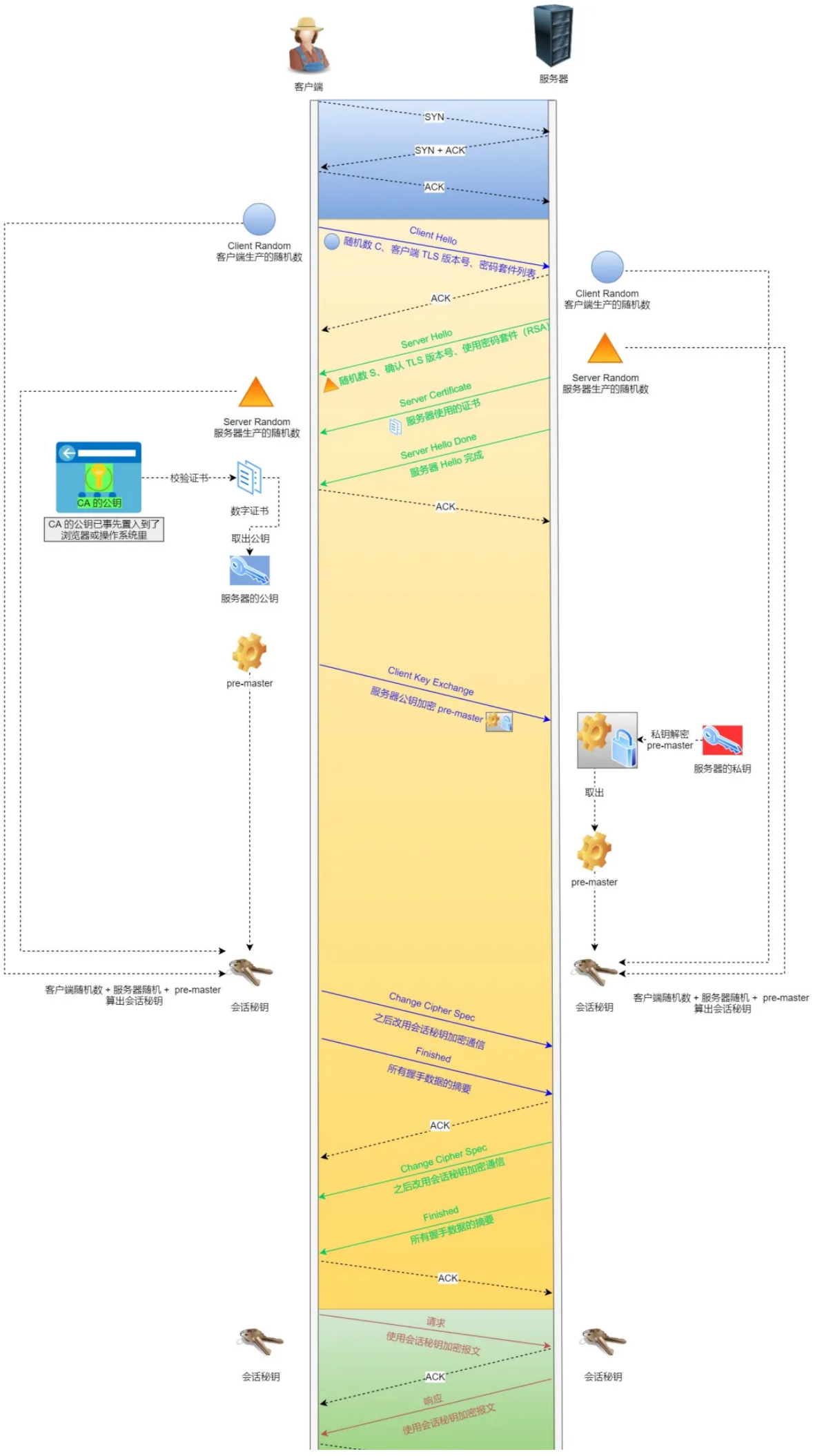
前两步也就是 SSL/TLS 的建立过程，也就是握手阶段。



上图简要概述来 TLS 的握手过程，其中每一个「框」都是一个记录（*record*），记录是 TLS 收发数据的基本单位，类似于 TCP 里的 segment。多个记录可以组合成一个 TCP 包发送，所以**通常经过「四个消息」就可以完成 TLS 握手，也就是需要 2个 RTT 的时延**，然后就可以在安全的通信环境里发送 HTTP 报文，实现 HTTPS 协议。

所以可以发现，HTTPS 是应用层协议，需要先完成 TCP 连接建立，然后走 TLS 握手过程后，才能建立通信安全的连接。

SSL/TLS 的「握手阶段」涉及**四次**通信，可见下图：


客户端
服务器
SYN
SYN + ACK
ACK
Client Hello
随机数 C、客户端 TLS 版本号、密码套件列表
ACK
Server Hello
随机数 S、确认 TLS 版本号、使用密码套件（RSA）
Server Certificate
服务器使用的证书
Server Hello Done
服务器 Hello 完成
ACK
Client Random
客户端生产的随机数
Server Random
服务器生产的随机数
CA 的公钥
校验证书
数字证书
取出公钥
CA 的公钥已事先置入到了浏览器或操作系统里
服务器的公钥
pre-master
Client Key Exchange
服务器公钥加密 pre-master
私钥解密 pre-master
服务器的私钥
取出
客户端随机数 + 服务器随机 + pre-master
算出会话秘钥
会话秘钥
Change Cipher Spec
之后改用会话秘钥加密通信
Finished
所有握手数据的摘要
ACK
Change Cipher Spec
之后改用会话秘钥加密通信
Finished
所有握手数据的摘要
ACK
请求
使用会话秘钥加密报文
ACK
响应
使用会话秘钥加密报文

SSL/TLS 协议建立的详细流程

第一次握手：ClientHello

首先，由客户端向服务器发起加密通信请求，也就是 `ClientHello` 请求。

在这一步，客户端主要向服务器发送以下信息：

- 客户端支持的 **SSL/TLS 协议版本**，如 TLS 1.2 版本。
- 客户端生产的随机数（**Client Random**），后面用于生产「会话秘钥」。
- 客户端支持的**密码套件列表**，如 RSA 加密算法。

第二次握手：ServerHello

服务器收到客户端请求后，向客户端发出响应，也就是 `SeverHello`。

服务器回应的内容有如下内容：

- 确认 **SSL/ TLS 协议版本**，如果浏览器不支持，则关闭加密通信。
- 服务器生产的随机数（**Server Random**），后面用于生产「会话秘钥」。
- **确认的密码套件列表**，如 RSA 加密算法。
- **服务器的数字证书**
- **Server Hello Done**：服务端握手结束通知

第三次握手：客户端回应

- 客户端验证服务器的数字证书。
- 若证书没有问题，客户端会从数字证书中取出服务器的公钥，然后使用它加密报文，向服务器发送如下信息：
  - 一个随机数（**pre-master key**），该随机数会被服务器公钥加密。
- 服务端收到后，用私钥解密，得到客户端发来的随机数 (pre-master)。
- 至此，**客户端和服务端共享了三个随机数，分别是 Client Random、Server Random、pre-master。**
- 双方根据已经得到的三个随机数，生成**会话密钥（Master Secret）**，它是对称密钥，用于对后续的 HTTP 请求/响应的数据加解密。
- 生成密钥后，客户端发送**加密通信算法改变通知（Change Cipher Spec）**，表示随后的信息都将用会话秘钥加密通信。
- **握手结束通知（Encrypted Handshake Message（Finishd））**，表示客户端的握手阶段已经结束。这一项同时把之前所有内容的发生的数据做个摘要，用来供服务端校验。

前两次握手明文，第三次握手中客户端发送的随机数使用公钥加密，服务端用私钥解密，再与前两次的随机数一同计算出对称密钥。之后全是对称密钥加密的密文。

第四次握手：服务器的最后回应

服务器收到客户端的信息后用私钥解密得到第三个随机数（`pre-master key`），再通过协商的加密算法，计算出本次通信的「会话秘钥」。然后，向客户端发生最后的信息：

（1）**加密通信算法改变通知（Change Cipher Spec）**，表示随后的信息都将用「会话秘钥」加密通信。

（2）**握手结束通知（Encrypted Handshake Message（Finishd））**，表示服务器的握手阶段已经结束。这一项同时把之前所有内容的发生的数据做个摘要，用来供客户端校验。

至此，整个 SSL/TLS 的握手阶段全部结束。接下来，客户端与服务器进入加密通信，就完全是使用普通的 HTTP 协议，只不过用「会话秘钥」加密内容。