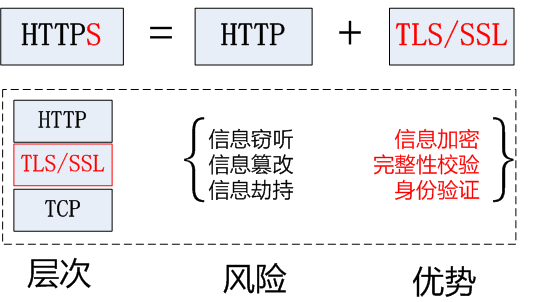
**HTTPS加密协议详解(一)：HTTPS基础知识**

**HTTPS基础知识：**HTTPS (Secure Hypertext Transfer Protocol)安全超文本传输协议，是一个安全通信通道，它基于HTTP开发用于在客户计算机和服务器之间交换信息。它使用安全套接字层([SSL](https://www.wosign.com/basic/howsslwork.htm))进行信息交换，简单来说它是HTTP的安全版,是使用TLS/SSL加密的HTTP协议。

HTTP协议采用明文传输信息，存在信息窃听、信息篡改和信息劫持的风险，而协议TLS/SSL具有身份验证、信息加密和完整性校验的功能，可以避免此类问题发生。

TLS/SSL全称安全传输层协议Transport Layer Security, 是介于TCP和HTTP之间的一层安全协议，不影响原有的TCP协议和HTTP协议，所以使用[HTTPS](https://www.wosign.com/)基本上不需要对HTTP页面进行太多的改造。



**HTTPS和HTTP的区别是什么？**

广泛应用于互联网世界的HTTP想必是大家再熟悉不过的了，然而细心的朋友可能发现淘宝、百度、网上银行等网站都变成HTTPS开头，并且还有一把小绿锁挂在地址栏，那么**HTTPS和HTTP的区别是什么**呢?



**一、什么是HTTPS**

HTTPS是在HTTP上建立[SSL加密](https://www.wosign.com/)层，并对传输数据进行加密，是HTTP协议的安全版。HTTPS主要作用是：

（1）对数据进行加密，并建立一个信息安全通道，来保证传输过程中的数据安全;

（2）对网站服务器进行真实身份认证。

**二、什么是HTTP**

HTTP是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准(TCP)，用于从WWW服务器传输超文本到本地浏览器的传输协议。HTTP是采用明文形式进行数据传输，极易被不法份子窃取和篡改。

**三、HTTPS和HTTP的区别是什么**

1、HTTPS是加密传输协议，HTTP是名文传输协议;

2、HTTPS需要用到SSL证书，而HTTP不用;

3、HTTPS比HTTP更加安全，对搜索引擎更友好，利于SEO,参考：

（1）[为保护用户隐私安全,谷歌优先索引HTTPS网页](https://www.wosign.com/News/2015-1225-01.htm)

（2）[百度开放收录https站点，https全网化势不可挡](https://www.wosign.com/News/baidu-https.html)

4、 HTTPS标准端口443，HTTP标准端口80;

5、 HTTPS基于传输层，HTTP基于应用层;

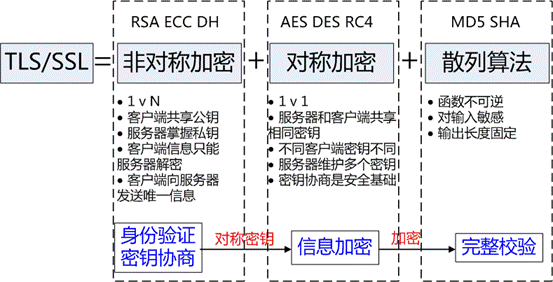
6、 HTTPS在浏览器显示绿色安全锁，HTTP没有显示;

总的来说HTTPS比HTTP更加安全，能够有效的保护网站用户的隐私信息安全，这也是为什么现在的HTTPS网站越来越多。如果不想你的网站因为数据泄露上头条的话，就赶快去沃通CA申请一张SSL证书为自己的网站实现HTTPS加密吧!

**HTTPS加密协议详解(二)：TLS/SSL工作原理**

HTTPS协议的主要功能基本都依赖于TLS/SSL协议，本节分析**TLS/SSL协议工作原理**。

TLS/SSL的功能实现主要依赖于三类基本算法：散列函数 Hash、对称加密和非对称加密，其利用非对称加密实现身份认证和密钥协商，对称加密算法采用协商的密钥对数据加密，基于散列函数验证信息的完整性。



**散列函数Hash**

常见的有 MD5、SHA1、SHA256，该类函数特点是函数单向不可逆、对输入非常敏感、输出长度固定，针对数据的任何修改都会改变散列函数的结果，用于防止信息篡改并验证数据的完整性;

在信息传输过程中，散列函数不能单独实现信息防篡改，因为明文传输，中间人可以修改信息之后重新计算信息摘要，因此需要对传输的信息以及信息摘要进行加密;

**对称加密**

常见的有 AES-CBC、DES、3DES、AES-GCM等，相同的密钥可以用于信息的加密和解密，掌握密钥才能获取信息，能够防止信息窃听，通信方式是1对1;

对称加密的优势是信息传输1对1，需要共享相同的密码，密码的安全是保证信息安全的基础，服务器和 N 个客户端通信，需要维持 N 个密码记录，且缺少修改密码的机制;

**非对称加密**

即常见的 RSA 算法，还包括 ECC、DH 等算法，算法特点是，密钥成对出现，一般称为公钥(公开)和私钥(保密)，公钥加密的信息只能私钥解开，私钥加密的信息只能公钥解开。因此掌握公钥的不同客户端之间不能互相解密信息，只能和掌握私钥的服务器进行加密通信，服务器可以实现1对多的通信，客户端也可以用来验证掌握私钥的服务器身份。

非对称加密的特点是信息传输1对多，服务器只需要维持一个私钥就能够和多个客户端进行加密通信，但服务器发出的信息能够被所有的客户端解密，且该算法的计算复杂，加密速度慢。

结合三类算法的特点，TLS的基本工作方式是，客户端使用非对称加密与服务器进行通信，实现身份验证并协商对称加密使用的密钥，然后对称加密算法采用协商密钥对信息以及信息摘要进行加密通信，不同的节点之间采用的对称密钥不同，从而可以保证信息只能通信双方获取。

**HTTPS加密协议详解(三)：PKI 体系**

**1、RSA身份验证的隐患**

身份验证和密钥协商是TLS的基础功能，要求的前提是合法的服务器掌握着对应的私钥。但RSA算法无法确保服务器身份的合法性，因为公钥并不包含服务器的信息，存在安全隐患:

客户端C和服务器S进行通信，中间节点M截获了二者的通信;

节点M自己计算产生一对公钥pub\_M和私钥pri\_M;

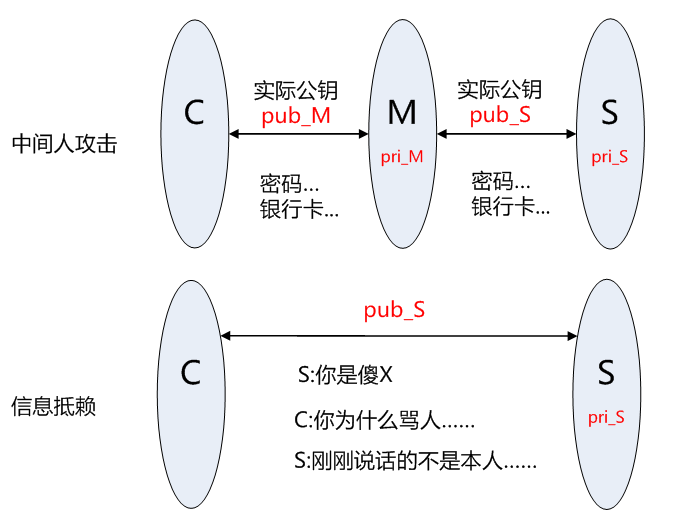
C向S请求公钥时，M把自己的公钥pub\_M发给了C;

C使用公钥 pub\_M加密的数据能够被M解密，因为M掌握对应的私钥pri\_M，而 C无法根据公钥信息判断服务器的身份，从而 C和 M之间建立了"可信"加密连接;

中间节点 M和服务器S之间再建立合法的连接，因此 C和 S之间通信被M完全掌握，M可以进行信息的窃听、篡改等操作。

另外，服务器也可以对自己的发出的信息进行否认，不承认相关信息是自己发出。

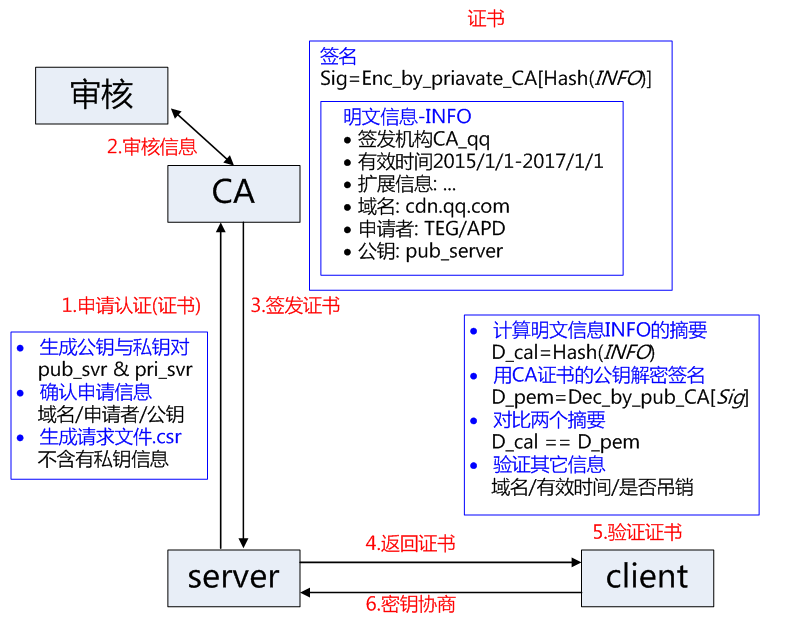
因此该方案下至少存在两类问题：中间人攻击和信息抵赖。



**2、身份验证CA和证书**

解决上述身份验证问题的关键是确保获取的公钥途径是合法的，能够验证服务器的身份信息，为此需要引入权威的第三方机构CA(如[沃通CA](https://www.wosign.com/))。CA 负责核实公钥的拥有者的信息，并颁发认证"证书"，同时能够为使用者提供证书验证服务，即**PKI体系**([PKI基础知识](https://www.wosign.com/basic/aboutPKI.htm))。

基本的原理为，CA负责审核信息，然后对关键信息利用私钥进行"签名"，公开对应的公钥，客户端可以利用公钥验证签名。CA也可以吊销已经签发的证书，基本的方式包括两类 CRL 文件和 OCSP。CA使用具体的流程如下：



a.服务方S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;

b.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;

c.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;

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;

d.客户端 C 向服务器 S 发出请求时，S 返回证书文件;

e.客户端 C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，即公钥合法;

f.客户端然后验证证书相关的域名信息、有效时间等信息;

g.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。

**在这个过程注意几点：**

a.申请证书不需要提供私钥，确保私钥永远只能服务器掌握;

b.证书的合法性仍然依赖于非对称加密算法，证书主要是增加了服务器信息以及签名;

c.内置 CA 对应的证书称为根证书，颁发者和使用者相同，自己为自己签名，即自签名证书（[为什么说"部署自签SSL证书非常不安全"](https://www.wosign.com/FAQ/selfsigned_SSL_insecure.htm)）

d.证书=公钥+申请者与颁发者信息+签名;

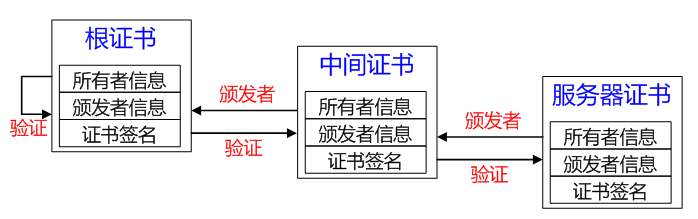
**3、证书链**

如 CA根证书和服务器证书中间增加一级证书机构，即中间证书，证书的产生和验证原理不变，只是增加一层验证，只要最后能够被任何信任的CA根证书验证合法即可。

a.服务器证书 server.pem 的签发者为中间证书机构 inter，inter 根据证书 inter.pem 验证 server.pem 确实为自己签发的有效证书;

b.中间证书 inter.pem 的签发 CA 为 root，root 根据证书 root.pem 验证 inter.pem 为自己签发的合法证书;

c.客户端内置信任 CA 的 root.pem 证书，因此服务器证书 server.pem 的被信任。



服务器证书、中间证书与根证书在一起组合成一条合法的[证书链](https://www.wosign.com/Support/ssl_Chain.htm)，证书链的验证是自下而上的信任传递的过程。

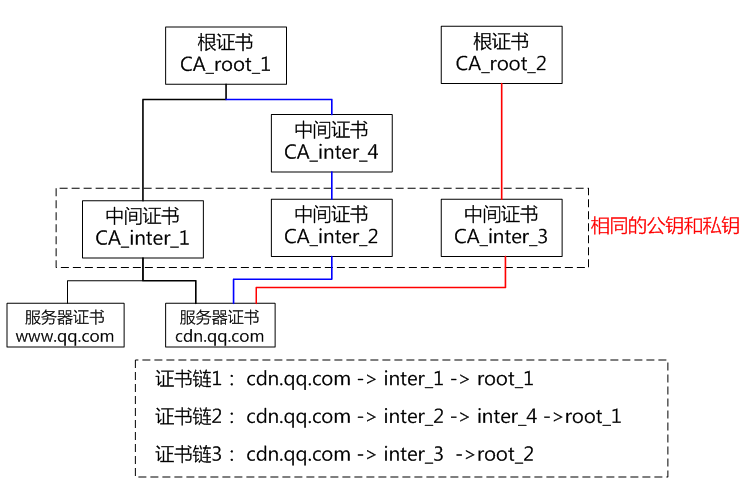
**二级证书结构存在的优势：**

a.减少根证书结构的管理工作量，可以更高效的进行证书的审核与签发;

b.根证书一般内置在客户端中，私钥一般离线存储，一旦私钥泄露，则吊销过程非常困难，无法及时补救;

c.中间证书结构的私钥泄露，则可以快速在线吊销，并重新为用户签发新的证书;

d.证书链四级以内一般不会对 HTTPS 的性能造成明显影响。



**证书链有以下特点：**

a.同一本服务器证书可能存在多条合法的证书链。

因为证书的生成和验证基础是公钥和私钥对，如果采用相同的公钥和私钥生成不同的中间证书，针对被签发者而言，该签发机构都是合法的 CA，不同的是中间证书的签发机构不同;

b.不同证书链的层级不一定相同，可能二级、三级或四级证书链。

中间证书的签发机构可能是根证书机构也可能是另一个中间证书机构，所以证书链层级不一定相同。

**4、证书吊销**

CA 机构能够签发证书，同样也存在机制宣布以往签发的证书无效。证书使用者不合法，CA 需要废弃该证书;或者私钥丢失，使用者申请让证书无效。主要存在两类机制：CRL 与 OCSP。

a.CRL

Certificate Revocation List, 证书吊销列表([什么是证书吊销列表(CRL)？吊销列表起什么作用](https://www.wosign.com/FAQ/CRL_OCSP.htm))，一个单独的文件。该文件包含了 CA 已经吊销的证书序列号(唯一)与吊销日期，同时该文件包含生效日期并通知下次更新该文件的时间，当然该文件必然包含 CA 私钥的签名以验证文件的合法性。

证书中一般会包含一个 URL 地址 CRL Distribution Point，通知使用者去哪里下载对应的 CRL 以校验证书是否吊销。该吊销方式的优点是不需要频繁更新，但是不能及时吊销证书，因为 CRL 更新时间一般是几天，这期间可能已经造成了极大损失。

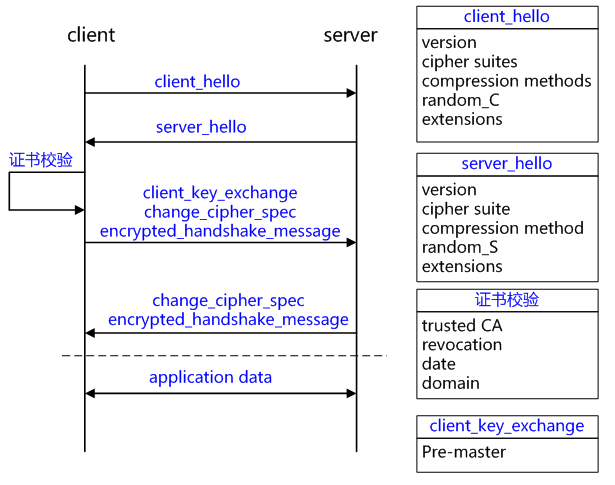
b.OCSP

Online Certificate Status Protocol, 证书状态在线查询协议，一个实时查询证书是否吊销的方式。请求者发送证书的信息并请求查询，服务器返回正常、吊销或未知中的任何一个状态。证书中一般也会包含一个 OCSP 的 URL 地址，要求查询服务器具有良好的性能。部分 CA 或大部分的自签 CA (根证书)都是未提供 CRL 或 OCSP 地址的，对于吊销证书会是一件非常麻烦的事情。

**HTTPS加密协议详解(四)：TLS/SSL握手过程**

**1、握手与密钥协商过程**

基于RSA握手和密钥交换的客户端验证服务器为示例详解**TLS/SSL握手过程**。



(1).client\_hello

客户端发起请求，以明文传输请求信息，包含版本信息，加密套件候选列表，压缩算法候选列表，随机数，扩展字段等信息，相关信息如下：

支持的最高TSL协议版本version，从低到高依次 SSLv2 SSLv3 TLSv1 TLSv1.1 TLSv1.2，当前基本不再使用低于 TLSv1 的版本;

客户端支持的加密套件 cipher suites 列表， 每个加密套件对应前面 TLS 原理中的四个功能的组合：认证算法 Au (身份验证)、密钥交换算法 KeyExchange(密钥协商)、对称加密算法 Enc (信息加密)和信息摘要 Mac(完整性校验);

支持的压缩算法 compression methods 列表，用于后续的信息压缩传输;

随机数 random\_C，用于后续的密钥的生成;

扩展字段 extensions，支持协议与算法的相关参数以及其它辅助信息等，常见的 SNI 就属于扩展字段，后续单独讨论该字段作用。

(2).server\_hello+server\_certificate+sever\_hello\_done

(a) server\_hello, 服务端返回协商的信息结果，包括选择使用的协议版本 version，选择的加密套件 cipher suite，选择的压缩算法 compression method、随机数 random\_S 等，其中随机数用于后续的密钥协商;

(b)server\_certificates, 服务器端配置对应的证书链，用于身份验证与密钥交换;

(c) server\_hello\_done，通知客户端 server\_hello 信息发送结束;

(3).证书校验

客户端验证证书的合法性，如果验证通过才会进行后续通信，否则根据错误情况不同做出提示和操作，合法性验证包括如下：

证书链的可信性 trusted certificate path，方法如前文所述;

证书是否吊销 revocation，有两类方式离线 CRL 与在线 OCSP，不同的客户端行为会不同;

有效期 expiry date，证书是否在有效时间范围;

域名 domain，核查证书域名是否与当前的访问域名匹配，匹配规则后续分析;

(4).client\_key\_exchange+change\_cipher\_spec+encrypted\_handshake\_message

(a) client\_key\_exchange，合法性验证通过之后，客户端计算产生随机数字 Pre-master，并用证书公钥加密，发送给服务器;

(b) 此时客户端已经获取全部的计算协商密钥需要的信息：两个明文随机数 random\_C 和 random\_S 与自己计算产生的 Pre-master，计算得到协商密钥;

enc\_key=Fuc(random\_C, random\_S, Pre-Master)

(c) change\_cipher\_spec，客户端通知服务器后续的通信都采用协商的通信密钥和加密算法进行加密通信;

(d) encrypted\_handshake\_message，结合之前所有通信参数的 hash 值与其它相关信息生成一段数据，采用协商密钥 session secret 与算法进行加密，然后发送给服务器用于数据与握手验证;

(5).change\_cipher\_spec+encrypted\_handshake\_message

(a) 服务器用私钥解密加密的 Pre-master 数据，基于之前交换的两个明文随机数 random\_C 和 random\_S，计算得到协商密钥:enc\_key=Fuc(random\_C, random\_S, Pre-Master);

(b) 计算之前所有接收信息的 hash 值，然后解密客户端发送的 encrypted\_handshake\_message，验证数据和密钥正确性;

(c) change\_cipher\_spec, 验证通过之后，服务器同样发送 change\_cipher\_spec 以告知客户端后续的通信都采用协商的密钥与算法进行加密通信;

(d) encrypted\_handshake\_message, 服务器也结合所有当前的通信参数信息生成一段数据并采用协商密钥 session secret 与算法加密并发送到客户端;

(6).握手结束

客户端计算所有接收信息的 hash 值，并采用协商密钥解密 encrypted\_handshake\_message，验证服务器发送的数据和密钥，验证通过则握手完成;

(7).加密通信

开始使用协商密钥与算法进行加密通信。

**注意：**

(a) 服务器也可以要求验证客户端，即[双向认证](https://www.wosign.com/FAQ/faq2016-0216-01.htm)，可以在过程2要发送 client\_certificate\_request 信息，客户端在过程4中先发送 client\_certificate与certificate\_verify\_message 信息，证书的验证方式基本相同，certificate\_verify\_message 是采用client的私钥加密的一段基于已经协商的通信信息得到数据，服务器可以采用对应的公钥解密并验证;

(b) 根据使用的密钥交换算法的不同，如 [ECC](https://www.wosign.com/News/About_ECC.htm) 等，协商细节略有不同，总体相似;

(c) sever key exchange 的作用是 server certificate 没有携带足够的信息时，发送给客户端以计算 pre-master，如基于 DH 的证书，公钥不被证书中包含，需要单独发送;

(d) change cipher spec 实际可用于通知对端改版当前使用的加密通信方式，当前没有深入解析;

(e) alter message 用于指明在握手或通信过程中的状态改变或错误信息，一般告警信息触发条件是连接关闭，收到不合法的信息，信息解密失败，用户取消操作等，收到告警信息之后，通信会被断开或者由接收方决定是否断开连接。

**2、会话缓存握手过程**

为了加快建立握手的速度，减少协议带来的性能降低和资源消耗(具体分析在后文)，TLS 协议有两类会话缓存机制：会话标识 session ID 与会话记录 session ticket。

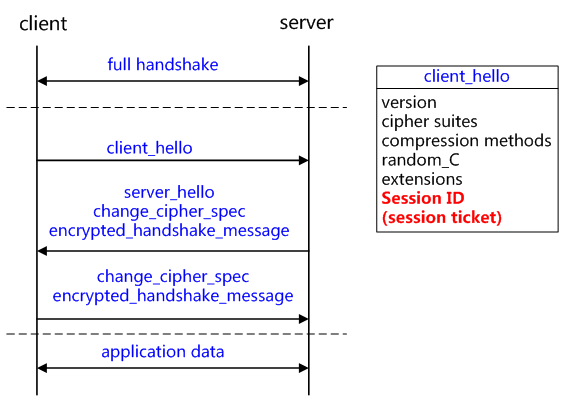
session ID 由服务器端支持，协议中的标准字段，因此基本所有服务器都支持，服务器端保存会话ID以及协商的通信信息，Nginx 中1M 内存约可以保存4000个 session ID 机器相关信息，占用服务器资源较多;

session ticket 需要服务器和客户端都支持，属于一个扩展字段，支持范围约60%(无可靠统计与来源)，将协商的通信信息加密之后发送给客户端保存，密钥只有服务器知道，占用服务器资源很少。

二者对比，主要是保存协商信息的位置与方式不同，类似与 http 中的 session 与 cookie。

二者都存在的情况下，(nginx 实现)优先使用 session\_ticket。

握手过程如下图：



**注意：**虽然握手过程有1.5个来回，但是最后客户端向服务器发送的第一条应用数据不需要等待服务器返回的信息，因此握手延时是1\*RTT。

(1).会话标识 session ID

(a) 如果客户端和服务器之间曾经建立了连接，服务器会在握手成功后返回 session ID，并保存对应的通信参数在服务器中;

(b) 如果客户端再次需要和该服务器建立连接，则在 client\_hello 中 session ID 中携带记录的信息，发送给服务器;

(c) 服务器根据收到的 session ID 检索缓存记录，如果没有检索到货缓存过期，则按照正常的握手过程进行;

(d) 如果检索到对应的缓存记录，则返回 change\_cipher\_spec 与 encrypted\_handshake\_message 信息，两个信息作用类似，encrypted\_handshake\_message 是到当前的通信参数与 master\_secret的hash 值;

(f) 如果客户端能够验证通过服务器加密数据，则客户端同样发送 change\_cipher\_spec 与 encrypted\_handshake\_message 信息;

(g) 服务器验证数据通过，则握手建立成功，开始进行正常的加密数据通信。

(2).会话记录 session ticket

(a) 如果客户端和服务器之间曾经建立了连接，服务器会在 new\_session\_ticket 数据中携带加密的 session\_ticket 信息，客户端保存;

(b) 如果客户端再次需要和该服务器建立连接，则在 client\_hello 中扩展字段 session\_ticket 中携带加密信息，一起发送给服务器;

(c) 服务器解密 sesssion\_ticket 数据，如果能够解密失败，则按照正常的握手过程进行;

(d) 如果解密成功，则返回 change\_cipher\_spec 与 encrypted\_handshake\_message 信息，两个信息作用与 session ID 中类似;

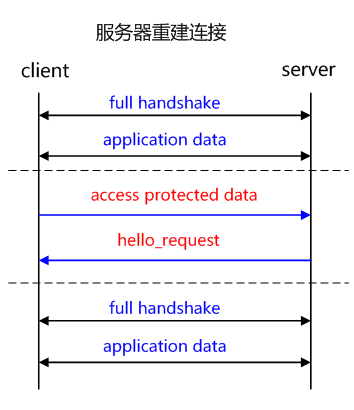
(f)如果客户端能够验证通过服务器加密数据，则客户端同样发送 change\_cipher\_spec与encrypted\_handshake\_message 信息;

(g) 服务器验证数据通过，则握手建立成功，开始进行正常的加密数据通信。

**3、重建连接**

重建连接 renegotiation 即放弃正在使用的 TLS 连接，从新进行身份认证和密钥协商的过程，特点是不需要断开当前的数据传输就可以重新身份认证、更新密钥或算法，因此服务器端存储和缓存的信息都可以保持。客户端和服务器都能够发起重建连接的过程，当前 windows 2000 & XP 与 SSL 2.0不支持。

(1).服务器重建连接



服务器端重建连接一般情况是客户端访问受保护的数据时发生。基本过程如下：

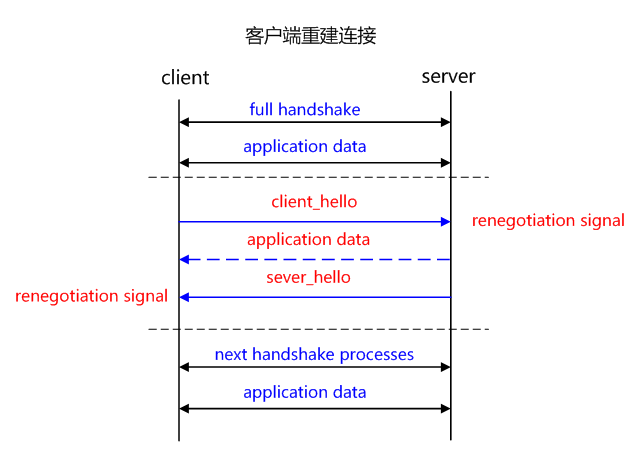
(a) 客户端和服务器之间建立了有效 TLS 连接并通信;

(b) 客户端访问受保护的信息;

(c) 服务器端返回 hello\_request 信息;

(d) 客户端收到 hello\_request 信息之后发送 client\_hello 信息，开始重新建立连接。

(2).客户端重建连接



客户端重建连接一般是为了更新通信密钥。

(a) 客户端和服务器之间建立了有效 TLS 连接并通信;

(b) 客户端需要更新密钥，主动发出 client\_hello 信息;

(c) 服务器端收到 client\_hello 信息之后无法立即识别出该信息非应用数据，因此会提交给下一步处理，处理完之后会返回通知该信息为要求重建连接;

(d) 在确定重建连接之前，服务器不会立即停止向客户端发送数据，可能恰好同时或有缓存数据需要发送给客户端，但是客户端不会再发送任何信息给服务器;

(e) 服务器识别出重建连接请求之后，发送 server\_hello 信息至客户端;

(f) 客户端也同样无法立即判断出该信息非应用数据，同样提交给下一步处理，处理之后会返回通知该信息为要求重建连接;

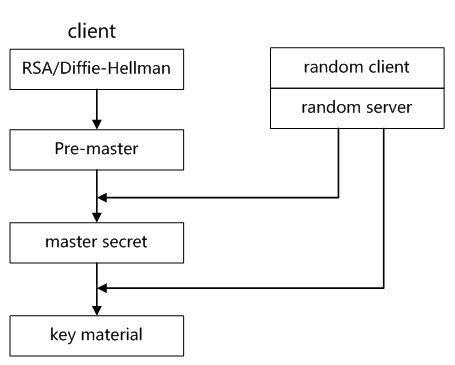
(g) 客户端和服务器开始新的重建连接的过程。

**4、密钥计算**

上节提到了两个明文传输的随机数 random\_C 和 random\_S 与通过加密在服务器和客户端之间交换的 Pre-master，三个参数作为密钥协商的基础。本节讨论说明密钥协商的基本计算过程以及通信过程中的密钥使用。

(1).计算 Key

涉及参数 random client 和 random server, Pre-master, Master secret, key material, 计算密钥时，服务器和客户端都具有这些基本信息，交换方式在上节中有说明，计算流程如下：



(a) 客户端采用 RSA 或 Diffie-Hellman 等加密算法生成 Pre-master;

(b) Pre-master 结合 random client 和 random server 两个随机数通过 PseudoRandomFunction(PRF)计算得到 Master secret;

(c) Master secret 结合 random client 和 random server 两个随机数通过迭代计算得到 Key material;

以下为一些重要的记录，可以解决部分爱深入研究朋友的疑惑，copy的材料，分享给大家：

(a) PreMaster secret 前两个字节是 TLS 的版本号，这是一个比较重要的用来核对握手数据的版本号，因为在 Client Hello 阶段，客户端会发送一份加密套件列表和当前支持的 SSL/TLS 的版本号给服务端，而且是使用明文传送的，如果握手的数据包被破解之后，攻击者很有可能串改数据包，选择一个安全性较低的加密套件和版本给服务端，从而对数据进行破解。所以，服务端需要对密文中解密出来对的 PreMaster 版本号跟之前 Client Hello 阶段的版本号进行对比，如果版本号变低，则说明被串改，则立即停止发送任何消息。(copy)

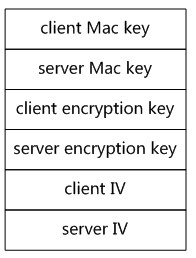
(b) 不管是客户端还是服务器，都需要随机数，这样生成的密钥才不会每次都一样。由于 SSL 协议中证书是静态的，因此十分有必要引入一种随机因素来保证协商出来的密钥的随机性。

对于 RSA 密钥交换算法来说，pre-master-key 本身就是一个随机数，再加上 hello 消息中的随机，三个随机数通过一个密钥导出器最终导出一个对称密钥。

pre master 的存在在于 SSL 协议不信任每个主机都能产生完全随机的随机数，如果随机数不随机，那么 pre master secret 就有可能被猜出来，那么仅适用 pre master secret 作为密钥就不合适了，因此必须引入新的随机因素，那么客户端和服务器加上 pre master secret 三个随机数一同生成的密钥就不容易被猜出了，一个伪随机可能完全不随机，可是三个伪随机就十分接近随机了，每增加一个自由度，随机性增加的可不是一。

(2).密钥使用

Key 经过12轮迭代计算会获取到12个 hash 值，分组成为6个元素，列表如下：



(a) mac key、encryption key 和 IV 是一组加密元素，分别被客户端和服务器使用，但是这两组元素都被两边同时获取;

(b) 客户端使用 client 组元素加密数据，服务器使用 client 元素解密;服务器使用 server 元素加密，client 使用 server 元素解密;

(c) 双向通信的不同方向使用的密钥不同，破解通信至少需要破解两次;

(d) encryption key 用于对称加密数据;

(e) IV 作为很多加密算法的初始化向量使用，具体可以研究对称加密算法;

(f) Mac key 用于数据的完整性校验;

（3）.数据加密通信过程

(a) 对应用层数据进行分片成合适的 block;

(b) 为分片数据编号，防止重放攻击;

(c) 使用协商的压缩算法压缩数据;

(d) 计算 MAC 值和压缩数据组成传输数据;

(e) 使用 client encryption key 加密数据，发送给服务器 server;

(f) server 收到数据之后使用 client encrytion key 解密，校验数据，解压缩数据，重新组装。

注：MAC值的计算包括两个 Hash 值：client Mac key 和 Hash (编号、包类型、长度、压缩数据)。

**5、抓包分析**

关于抓包不再详细分析，按照前面的分析，基本的情况都能够匹配，根据平常定位问题的过程，个人提些认为需要注意的地方：

(1).抓包 HTTP 通信，能够清晰的看到通信的头部和信息的明文，但是 HTTPS 是加密通信，无法看到 HTTP 协议的相关头部和数据的明文信息，

(2).抓包 HTTPS 通信主要包括三个过程：TCP 建立连接、TLS 握手、TLS 加密通信，主要分析 HTTPS 通信的握手建立和状态等信息。

(3).client\_hello

根据 version 信息能够知道客户端支持的最高的协议版本号，如果是 SSL 3.0 或 TLS 1.0 等低版本协议，非常注意可能因为版本低引起一些握手失败的情况;

根据 extension 字段中的 server\_name 字段判断是否支持SNI，存在则支持，否则不支持，对于定位握手失败或证书返回错误非常有用;

会话标识 session ID 是标准协议部分，如果没有建立过连接则对应值为空，不为空则说明之前建立过对应的连接并缓存;

会话记录 session ticke t是扩展协议部分，存在该字段说明协议支持 sesssion ticket，否则不支持，存在且值为空，说明之前未建立并缓存连接，存在且值不为空，说明有缓存连接。

(4).server\_hello

根据 TLS version 字段能够推测出服务器支持的协议的最高版本，版本不同可能造成握手失败;

基于 cipher\_suite 信息判断出服务器优先支持的加密协议;

(5).ceritficate

服务器配置并返回的证书链，根据证书信息并于服务器配置文件对比，判断请求与期望是否一致，如果不一致，是否返回的默认证书。

(6).alert

告警信息 alert 会说明建立连接失败的原因即告警类型，对于定位问题非常重要。

**HTTPS加密协议详解(五)：HTTPS性能与优化**

**1、HTTPS性能损耗**

前文讨论了HTTPS原理与优势：身份验证、信息加密与完整性校验等，且未对TCP和HTTP协议做任何修改。但通过增加新协议以实现更安全的通信必然需要付出代价，HTTPS协议的性能损耗主要体现如下：

(1).增加延时

分析前面的握手过程，一次完整的握手至少需要两端依次来回两次通信，至少增加延时2\* RTT，利用会话缓存从而复用连接，延时也至少1\* RTT\*。

(2).消耗较多的CPU资源

除数据传输之外，HTTPS通信主要包括对对称加解密、非对称加解密(服务器主要采用私钥解密数据);压测 TS8 机型的单核 CPU：对称加密算法AES-CBC-256 吞吐量 600Mbps，非对称 RSA 私钥解密200次/s。不考虑其它软件层面的开销，10G 网卡为对称加密需要消耗 CPU 约17核，24核CPU最多接入 HTTPS 连接 4800;

静态节点当前10G 网卡的 TS8 机型的 HTTP 单机接入能力约为10w/s，如果将所有的HTTP连接变为HTTPS连接，则明显RSA的解密最先成为瓶颈。因此，RSA的解密能力是当前困扰HTTPS接入的主要难题。

**2、HTTPS接入优化**

(1).CDN接入

HTTPS 增加的延时主要是传输延时 RTT，RTT 的特点是节点越近延时越小，CDN 天然离用户最近，因此选择使用 CDN 作为 HTTPS 接入的入口，将能够极大减少接入延时。CDN 节点通过和业务服务器维持长连接、会话复用和链路质量优化等可控方法，极大减少 HTTPS 带来的延时。

(2).会话缓存

虽然前文提到 HTTPS 即使采用会话缓存也要至少1\*RTT的延时，但是至少延时已经减少为原来的一半，明显的延时优化;同时，基于会话缓存建立的 HTTPS 连接不需要服务器使用RSA私钥解密获取 Pre-master 信息，可以省去CPU 的消耗。如果业务访问连接集中，缓存命中率高，则HTTPS的接入能力讲明显提升。当前TRP平台的缓存命中率高峰时期大于30%，10k/s的接入资源实际可以承载13k/的接入，收效非常可观。

(3).硬件加速

为接入服务器安装专用的SSL硬件加速卡，作用类似 GPU，释放 CPU，能够具有更高的 HTTPS 接入能力且不影响业务程序的。测试某硬件加速卡单卡可以提供35k的解密能力，相当于175核 CPU，至少相当于7台24核的服务器，考虑到接入服务器其它程序的开销，一张硬件卡可以实现接近10台服务器的接入能力。

(4).远程解密

本地接入消耗过多的 CPU 资源，浪费了网卡和硬盘等资源，考虑将最消耗 CPU 资源的RSA解密计算任务转移到其它服务器，如此则可以充分发挥服务器的接入能力，充分利用带宽与网卡资源。远程解密服务器可以选择 CPU 负载较低的机器充当，实现机器资源复用，也可以是专门优化的高计算性能的服务器。当前也是 CDN 用于大规模HTTPS接入的解决方案之一。

(5).SPDY/HTTP2

前面的方法分别从减少传输延时和单机负载的方法提高 HTTPS 接入性能，但是方法都基于不改变 HTTP 协议的基础上提出的优化方法，SPDY/HTTP2 利用 TLS/SSL 带来的优势，通过修改协议的方法来提升 HTTPS 的性能，提高下载速度等。

**大型网站HTTPS实践(2)：HTTPS网站速度优化**

**1 前言**

[HTTPS](https://www.wosign.com/) 在保护用户隐私，防止流量劫持方面发挥着非常关键的作用，但与此同时，HTTPS 也会降低用户访问速度，增加网站服务器的计算资源消耗。本文主要介绍HTTPS对用户体验的影响。

**2 HTTPS对访问速度的影响**

在介绍HTTPS[速度优化](https://www.wosign.com/News/TLS13-HTTPS.htm)策略之前，先来看下 HTTPS 对速度有什么影响。影响主要来自两方面：

**·**协议交互所增加的网络 RTT(round trip time)。

**·**加解密相关的计算耗时。

下面分别介绍一下。

**2.1 网络耗时增加**

由于 HTTP 和 HTTPS 都需要 DNS 解析，并且大部分情况下使用了 DNS 缓存，为了突出对比效果，忽略主域名的 DNS 解析时间。

用户使用 HTTP 协议访问http://www.baidu.com(或者 www.baidu.com) 时会有如下网络上的交互耗时：

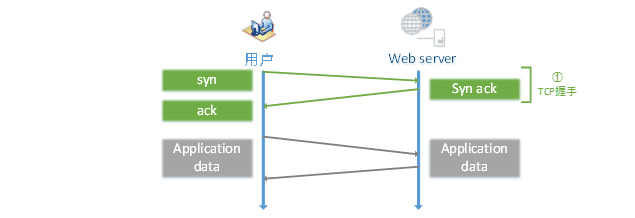


图 1 HTTP 首个请求的网络耗时

可见，用户只需要完成 TCP 三次握手建立 TCP 连接就能够直接发送 HTTP 请求获取应用层数据，此外在整个访问过程中也没有需要消耗计算资源的地方。

接下来看 HTTPS 的访问过程，相比 HTTP 要复杂很多，在部分场景下，使用 HTTPS 访问有可能增加 7 个 RTT。如下图：

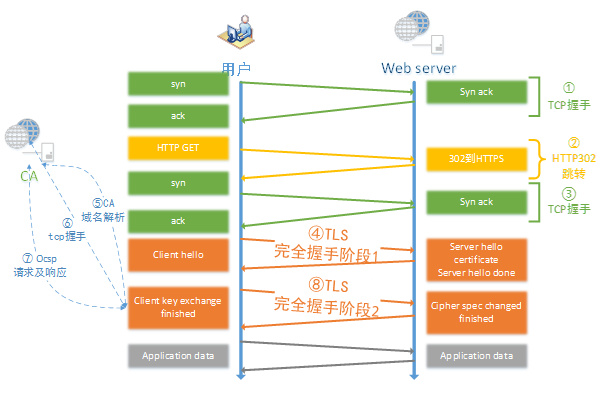


图 2 HTTPS 首次请求对访问速度的影响

HTTPS 首次请求需要的网络耗时解释如下：

1， 三次握手建立 TCP 连接。耗时一个 RTT。

2， 使用 HTTP 发起 GET 请求，服务端返回 302 跳转到 https://www.baidu.com。需要一个 RTT 以及 302 跳转延时。

a) 大部分情况下用户不会手动输入 https://www.baidu.com 来访问 HTTPS，服务端只能返回 302 强制浏览器跳转到 HTTPS。

b) 浏览器处理 302 跳转也需要耗时。

3， 三次握手重新建立 TCP 连接。耗时一个 RTT。

a) 302 跳转到 HTTPS 服务器之后，由于端口和服务器不同，需要重新完成三次握手，建立 TCP 连接。

4， TLS 完全握手阶段一。耗时至少一个 RTT。

a) 这个阶段主要是完成加密套件的协商和证书的身份认证。

b) 服务端和浏览器会协商出相同的密钥交换算法、对称加密算法、内容一致性校验算法、证书签名算法、椭圆曲线(非 ECC 算法不需要)等。

c) 浏览器获取到证书后需要校验证书的有效性，比如是否过期，是否撤销。

5， 解析 CA 站点的 DNS。耗时一个 RTT。

a) 浏览器获取到证书后，有可能需要发起 OCSP 或者 CRL 请求，查询证书状态。

b) 浏览器首先获取证书里的 CA 域名。

c) 如果没有命中缓存，浏览器需要解析 CA 域名的 DNS。

6， 三次握手建立 CA 站点的 TCP 连接。耗时一个 RTT。

a) DNS 解析到 IP 后，需要完成三次握手建立 TCP 连接。

7， 发起 OCSP 请求，获取响应。耗时一个 RTT。

8， 完全握手阶段二，耗时一个 RTT 及计算时间。

a) 完全握手阶段二主要是密钥协商。

9， 完全握手结束后，浏览器和服务器之间进行应用层(也就是 HTTP)数据传输。

当然不是每个请求都需要增加 7 个 RTT 才能完成 HTTPS 首次请求交互。大概只有不到 0.01% 的请求才有可能需要经历上述步骤，它们需要满足如下条件：

1， 必须是首次请求。即建立 TCP 连接后发起的第一个请求，该连接上的后续请求都不需要再发生上述行为。

2， 必须要发生完全握手，而正常情况下 80% 的请求能实现简化握手。

3， 浏览器需要开启 OCSP 或者 CRL 功能。Chrome 默认关闭了 ocsp 功能，firefox 和 IE 都默认开启。

4， 浏览器没有命中 OCSP 缓存。Ocsp 一般的更新周期是 7 天，firefox 的查询周期也是 7 天，也就说是 7 天中才会发生一次 ocsp 的查询。

5， 浏览器没有命中 CA 站点的 DNS 缓存。只有没命中 DNS 缓存的情况下才会解析 CA 的 DNS。

**2.2 计算耗时增加**

上节还只是简单描述了 HTTPS 关键路径上必须消耗的纯网络耗时，没有包括非常消耗 CPU 资源的计算耗时，事实上计算耗时也不小(30ms 以上)，从浏览器和服务器的角度分别介绍一下：

1， 浏览器计算耗时

a) RSA 证书签名校验，浏览器需要解密签名，计算证书哈希值。如果有多个证书链，浏览器需要校验多个证书。

b) RSA 密钥交换时，需要使用证书公钥加密 premaster。耗时比较小，但如果手机性能比较差，可能也需要 1ms 的时间。

c) ECC 密钥交换时，需要计算椭圆曲线的公私钥。

d) ECC 密钥交换时，需要使用证书公钥解密获取服务端发过来的 ECC 公钥。

e) ECC 密钥交换时，需要根据服务端公钥计算 master key。

f) 应用层数据对称加解密。

g) 应用层数据一致性校验。

2， 服务端计算耗时

a) RSA 密钥交换时需要使用证书私钥解密 premaster。这个过程非常消耗性能。

b) ECC 密钥交换时，需要计算椭圆曲线的公私钥。

c) ECC 密钥交换时，需要使用证书私钥加密 ECC 的公钥。

d) ECC 密钥交换时，需要根据浏览器公钥计算共享的 master key。

e) 应用层数据对称加解密。

f) 应用层数据一致性校验。

由于客户端的 CPU 和操作系统种类比较多，所以计算耗时不能一概而论。手机端的 HTTPS 计算会比较消耗性能，单纯计算增加的延迟至少在 50ms 以上。PC 端也会增加至少 10ms 以上的计算延迟。

服务器的性能一般比较强，但由于 RSA 证书私钥长度远大于客户端，所以服务端的计算延迟也会在 5ms 以上。

**3 结束语**

本系列的后续文章将进一步解释针对性的优化措施。

**转载-如何正确使用HTTPS加密协议经验分享(一)**

为了应对日益猖獗的流量劫持、数据泄漏、钓鱼欺诈等安全问题，越来越多的网站意识到实施[HTTPS加密的重要性](https://www.wosign.com/News/why-use-https.htm)，HTTPS 通过TLS 层和证书机制提供了内容加密、身份认证和数据完整性三大功能，可以有效防止数据被查看或篡改，以及防止中间人冒充。使用**HTTPS加密协议**替代HTTP明文协议，已经成为不争的共识。而如何正确应用HTTPS加密，成为大家更关心的话题。本文分享一些启用[HTTPS加密](https://www.wosign.com/)过程中的经验，重点是如何与一些新出的安全规范配合使用。

**一、理解Mixed Content**

HTTPS 网页中加载的HTTP资源被称之为Mixed Content(混合内容)，不同浏览器对Mixed Content 有不一样的处理规则。

**早期的IE**

早期的IE 在发现Mixed Content 请求时，会弹出「是否只查看安全传送的网页内容?」这样一个模态对话框，一旦用户选择「是」，所有Mixed Content 资源都不会加载;选择「否」，所有资源都加载。

**比较新的IE**

比较新的IE 将模态对话框改为页面底部的提示条，没有之前那么干扰用户。而且默认会加载图片类Mixed Content，其它如 JavaScript、CSS 等资源还是会根据用户选择来决定是否加载。

**现代浏览器**

现代浏览器(Chrome、Firefox、Safari、Microsoft Edge)，基本上都遵守了W3C 的Mixed Content 规范，将Mixed Content 分为Optionally-blockable 和Blockable 两类：

Optionally-blockable 类 Mixed Content 包含那些危险较小，即使被中间人篡改也无大碍的资源。现代浏览器默认会加载这类资源，同时会在控制台打印警告信息。这类资源包括：

·通过<img>标签加载的图片(包括 SVG 图片);

·通过<video> / <audio> 和<source>标签加载的视频或音频;

·预读的(Prefetched)资源;

除此之外所有的Mixed Content 都是 Blockable，浏览器必须禁止加载这类资源。所以现代浏览器中，对于HTTPS 页面中的JavaScript、CSS 等HTTP 资源，一律不加载，直接在控制台打印错误信息。

**移动浏览器**

前面所说都是桌面浏览器的行为，移动端情况比较复杂，当前大部分移动浏览器默认允许加载所有 Mixed Content。也就是说，对于移动浏览器来说，HTTPS 中的HTTP 资源，无论是图片还是 JavaScript、CSS，默认都会加载。

补充：上面这段结论源自于我大半年前的测试，本文评论中的ayanamist 同学反馈现状已经有所变化。我又做了一些测试，果然随着操作系统的升级，移动浏览器都开始遵循 Mixed Content 规范了。最新测试表明，对于Blockable 类 Mixed Content：

·iOS 9 以下的Safari，以及Android 5 以下的Webview，默认会加载;

·Android 各版本的Chrome，iOS 9+ 的 Safari，Android 5+ 的 Webview，默认不会加载;

一般选择了全站HTTPS，就要避免出现Mixed Content，页面所有资源请求都走HTTPS 协议才能保证所有平台所有浏览器下都没有问题。

**二、合理使用CSP**

CSP，全称是 Content Security Policy，它有非常多的指令，用来实现各种各样与页面内容安全相关的功能。这里只介绍两个与HTTPS 相关的指令，更多内容可以看我之前写的《[Content Security Policy Level 2 介绍](https://imququ.com/post/content-security-policy-level-2.html)》。

**block-all-mixed-content**

前面说过，对于HTTPS 中的图片等 Optionally-blockable 类HTTP 资源，现代浏览器默认会加载。图片类资源被劫持，通常不会有太大的问题，但也有一些风险，例如很多网页按钮是用图片实现的，中间人把这些图片改掉，也会干扰用户使用。

通过CSP的 block-all-mixed-content 指令，可以让页面进入对混合内容的严格检测(Strict Mixed Content Checking)模式。在这种模式下，所有非HTTPS 资源都不允许加载。跟其它所有CSP 规则一样，可以通过以下两种方式启用这个指令：

HTTP 响应头方式：

Content-Security-Policy: block-all-mixed-content

<meta>标签方式：

<meta http-equiv="Content-Security-Policy" content="block-all-mixed-content">

**upgrade-insecure-requests**

历史悠久的大站在往HTTPS迁移的过程中，工作量往往非常巨大，尤其是将所有资源都替换为HTTPS 这一步，很容易产生疏漏。即使所有代码都确认没有问题，很可能某些从数据库读取的字段中还存在HTTP链接。

而通过upgrade-insecure-requests 这个CSP 指令，可以让浏览器帮忙做这个转换。启用这个策略后，有两个变化：

· 页面所有 HTTP 资源，会被替换为HTTPS 地址再发起请求;

· 页面所有站内链接，点击后会被替换为HTTPS地址再跳转;

跟其它所有CSP 规则一样，这个指令也有两种方式来启用，具体格式请参考上一节。需要注意的是upgrade-insecure-requests只替换协议部分，所以只适用于HTTP/HTTPS 域名和路径完全一致的场景。

**三、合理使用HSTS**

在网站全站HTTPS 后，如果用户手动敲入网站的HTTP 地址，或者从其它地方点击了网站的HTTP链接，依赖于服务端301/302跳转才能使用HTTPS 服务。而第一次的HTTP 请求就有可能被劫持，导致请求无法到达服务器，从而构成 HTTPS 降级劫持。

**HSTS 基本使用**

这个问题可以通过 HSTS(HTTP Strict Transport Security，[RFC6797](https://tools.ietf.org/html/rfc6797))来解决。HSTS 是一个响应头，格式如下：

Strict-Transport-Security: max-age=expireTime [; includeSubDomains] [; preload]

**max-age**，单位是秒，用来告诉浏览器在指定时间内，这个网站必须通过HTTPS 协议来访问。也就是对于这个网站的HTTP 地址，浏览器需要先在本地替换为HTTPS 之后再发送请求。

**includeSubDomains**，可选参数，如果指定这个参数，表明这个网站所有子域名也必须通过HTTPS 协议来访问。

**preload**，可选参数，后面再介绍它的作用。

HSTS 这个响应头只能用于HTTPS 响应;网站必须使用默认的443 端口;必须使用域名，不能是IP。而且启用HSTS 之后，一旦网站证书错误，用户无法选择忽略。

**HSTS Preload List**

可以看到 HSTS 可以很好的解决HTTPS 降级攻击，但是对于HSTS 生效前的首次HTTP 请求，依然无法避免被劫持。浏览器厂商们为了解决这个问题，提出了HSTS Preload List 方案：内置一份可以定期更新的列表，对于列表中的域名，即使用户之前没有访问过，也会使用HTTPS 协议。

目前这个Preload List 由Google Chrome 维护，Chrome、Firefox、Safari、IE 11 和 Microsoft Edge 都在使用。如果要想把自己的域名加进这个列表，首先需要满足以下条件：

·拥有合法的证书(如果使用SHA-1 证书，过期时间必须早于2016 年);

·将所有 HTTP 流量重定向到HTTPS;

·确保所有子域名都启用了HTTPS;

·输出HSTS 响应头：max-age 不能低于18 周(10886400 秒);必须指定includeSubdomains 参数;必须指定preload 参数;

即便满足了上述所有条件，也不一定能进入HSTS Preload List。通过Chrome 的 chrome://net-internals/#hsts 工具，可以查询某个网站是否在Preload List 之中，还可以手动把某个域名加到本机 Preload List。

对于HSTS 以及HSTS Preload List，我的建议是只要你不能确保永远提供HTTPS 服务，就不要启用。因为一旦HSTS 生效，你再想把网站重定向为HTTP，之前的老用户会被无限重定向，唯一的办法是换新域名。

**四、CDN 安全**

对于大站来说，全站迁移到HTTPS 后还是得用CDN，只是必须选择支持HTTPS 的CDN 了。如果使用第三方CDN，安全方面有一些需要考虑的地方。

**五、合理使用SRI**

HTTPS 可以防止数据在传输中被篡改，合法的证书也可以起到验证服务器身份的作用，但是如果 CDN 服务器被入侵，导致静态文件在服务器上被篡改，HTTPS也无能为力。

W3C 的[SRI](https://www.w3.org/TR/SRI/)(Subresource Integrity)规范可以用来解决这个问题。SRI 通过在页面引用资源时指定资源的摘要签名，来实现让浏览器验证资源是否被篡改的目的。只要页面不被篡改，SRI 策略就是可靠的。

SRI 并不是HTTPS 专用，但如果主页面被劫持，攻击者可以轻松去掉资源摘要，从而失去浏览器的 SRI 校验机制。

**六、了解Keyless SSL**

另外一个问题是，在使用第三方CDN 的HTTPS 服务时，如果要使用自己的域名，需要把对应的证书私钥给第三方，这也是一件风险很高的事情。

Keyless SSL 技术让你可以不把证书私钥给第三方，改为提供一台实时计算的Key Server 即可。CDN 要用到私钥时，通过加密通道将必要的参数传给 Key Server，由 Key Server 算出结果并返回即可。整个过程中，私钥都保管在自己的 Key Server 之中，不会暴露给第三方。

好了，本文先就写到这里，需要注意的是本文提到的 CSP、HSTS 以及 SRI 等策略都只有最新的浏览器才支持，详细的支持度可以去[CanIUse](https://caniuse.com/)查。切换到HTTPS 之后，在性能优化上有很多新工作要做，这部分内容我在之前的博客中写过很多，这里不再重复，只说最重要的一点：既然都HTTPS了，赶紧上HTTP/2才是正道。

**转载-如何正确使用HTTPS加密协议经验分享(二)**

发布日期：2016-05-17

为什么同样是使用了[SSL证书](https://www.wosign.com/)，有些网站评分很低有些却很高?几天前，一位朋友问我：都说用[SSL Labs](https://wosign.ssllabs.com/)这个工具测试SSL安全性，为什么有些安全实力很强的大厂家评分也很低?我认为这个问题应该从两方面来看：

1)国内用户终端情况复杂，很多时候降低SSL安全配置是为了兼容更多用户;

2)确实有一些大厂家的SSL配置很不专业，尤其是配置了一些明显不该使用的CipherSuite。

我之前写的《[正确使用HTTPS加密协议经验分享(一)](https://www.wosign.com/News/https-protocols1.htm)》，主要介绍HTTPS加密协议如何与一些新出的安全规范配合使用，面向的是现代浏览器。而今天这篇文章，更多的是介绍启用HTTPS过程中在老旧浏览器下可能遇到的问题，以及如何取舍。

**SSL版本选择**

TLS(Transport Layer Security，传输层安全)的前身是SSL(Secure Sockets Layer，安全套接字层)，它最初的几个版本(SSL 1.0、SSL 2.0、SSL 3.0)由网景公司开发，从3.1 开始被IETF 标准化并改名，发展至今已经有TLS 1.0、TLS 1.1、TLS 1.2 三个版本。TLS 1.3 改动会比较大，目前还在草案阶段。

SSL 1.0 从未公开过，而SSL 2.0 和 SSL 3.0 都存在安全问题，不推荐使用。Nginx 从1.9.1 开始默认只支持TLS 的三个版本，以下是[Nginx 官方文档](https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_protocols)中对ssl\_protocols 配置的说明：

Syntax: ssl\_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];

Default: ssl\_protocols TLSv1 TLSv1.1 TLSv1.2;

Context: http, server

Enables the specified protocols. The TLSv1.1 and TLSv1.2 parameters work only when the OpenSSL library of version 1.0.1 or higher is used.

但不幸的是，IE 6 默认只支持SSLv2 和SSLv3([来源](https://www.ssllabs.com/ssltest/viewClient.html?name=IE&version=6&platform=XP))，也就是说HTTPS 网站要支持IE 6，就必须启用SSLv3。仅这一项就会导致SSL Labs 给出的评分降为C。

**加密套件选择**

加密套件(CipherSuite)，是在SSL 握手中需要协商的很重要的一个参数。客户端会在Client Hello 中带上它所支持的CipherSuite 列表，服务端会从中选定一个并通过Server Hello 返回。如果客户端支持的CipherSuite 列表与服务端配置的CipherSuite 列表没有交集，会导致无法完成协商，握手失败。

CipherSuite 包含多种技术，例如认证算法(Authentication)、加密算法(Encryption)、消息认证码算法(Message Authentication Code，简称为 MAC)、密钥交换算法(Key Exchange)和密钥衍生算法(Key Derivation Function)。

SSL 的CipherSuite 协商机制具有良好的扩展性，每个CipherSuite 都需要在IANA 注册，并被分配两个字节的标志。全部CipherSuite 可以在IANA 的TLS Cipher Suite Registry 页面查看。

OpenSSL库支持的全部CipherSuite 可以通过以下命令查看：

openssl ciphers -V | column -t

0xCC,0x14 - ECDHE-ECDSA-CHACHA20-POLY1305  TLSv1.2  Kx=ECDH  Au=ECDSA  Enc=ChaCha20-Poly1305  Mac=AEAD

... ...

0xCC,0x14 是CipherSuite 的编号，在SSL 握手中会用到。ECDHE-ECDSA-CHACHA20-POLY1305 是它的名称，之后几部分分别表示：用于TLSv1.2，使用ECDH 做密钥交换，使用ECDSA 做认证，使用ChaCha20-Poly1305 做对称加密，由于ChaCha20-Poly1305 是一种AEAD 模式，不需要MAC 算法，所以MAC 列显示为AEAD。

要了解CipherSuite 的更多内容，可以阅读这篇长文《[TLS 协议分析与现代加密通信协议设计](https://blog.helong.info/blog/2015/09/06/tls-protocol-analysis-and-crypto-protocol-design/)》。总之，在配置CipherSuite 时，请务必参考权威文档，如：[Mozilla的推荐配置](https://wiki.mozilla.org/Security/Server_Side_TLS#Recommended_configurations)、[CloudFlare使用的配置](https://github.com/cloudflare/sslconfig/blob/master/conf)。

以上Mozilla 文档中的「Old backward compatibility」配置，以及CloudFlare 的配置，都可以很好的兼容老旧浏览器，包括Windows XP / IE6。

之前见到某个大厂家居然支持包含EXPORT 的CipherSuite，这些套件在上世纪由于美国出口限制而被弱化过，已被攻破，实在没有理由再使用。

**SNI 扩展**

我们知道，在Nginx 中可以通过指定不同的server\_name来配置多个站点。HTTP/1.1 协议请求头中的Host 字段可以标识出当前请求属于哪个站点。但是对于HTTPS 网站来说，要想发送HTTP 数据，必须等待SSL 握手完成，而在握手阶段服务端就必须提供网站证书。对于在同一个IP 部署不同HTTPS 站点，并且还使用了不同证书的情况下，服务端怎么知道该发送哪个证书?

Server Name Indication，简称为SNI，是TLS 的一个扩展，为解决这个问题应运而生。有了SNI，服务端可以通过Client Hello 中的SNI 扩展拿到用户要访问网站的Server Name，进而发送与之匹配的证书，顺利完成SSL 握手。

Nginx在很早之前就支持了SNI，可以通过nginx -V 来验证。以下是我的验证结果：

./nginx -V

nginx version: nginx/1.9.9

built by gcc 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)

built with OpenSSL 1.0.2e-dev xx ‍ xxxx

TLS SNI support enabled

configure arguments: --with-openssl=../openssl --with-http\_ssl\_module --with-http\_v2\_module

然而，并不是所有浏览器都支持SNI，以下是常见浏览器支持SNI 的最低版本：



可以看到，现在还有一定用户量的Windows XP IE6~8、Android 2.x Webview 都不支持SNI。如果要避免在这些浏览器中出现证书错误，只能将使用不同证书的HTTPS 站点部署在不同IP 上，最简单的做法是分开部署到不同机器上。

另外，包含 SAN(Subject Alternative Name)的证书能同时支持多个域名，也可以用于解决这个问题。

**证书选择**

HTTPS 网站需要通过CA 取得合法证书，证书通过数字签名技术确保第三方无法伪造。证书的简单原理如下：

· 根据版本号、序列号、签名算法标识、发行者名称、有效期、证书主体名、证书主体公钥信息、发行商唯一标识、主体唯一标识、扩展生成 TBSCertificate(To Be Signed Certificate，待签名证书)信息;

· 签发数字签名：使用HASH 函数对TBSCertificate 计算得到消息摘要，用CA 的私钥对消息摘要进行加密，得到签名;

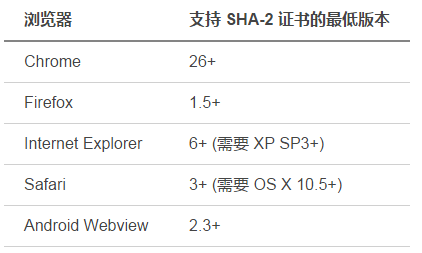
· 校验数字签名：使用相同的HASH 函数对TBSCertificate 计算得到消息摘要，与使用CA 公钥解密签名得到内容相比较;

使用SHA-1 做为HASH 函数的证书被称之为SHA-1 证书，由于目前已经找到SHA-1 的碰撞条件，将证书换成使用更安全的SHA-2 做为 HASH 函数的SHA-2 证书被提上日程。

实际上，微软已经宣称自2017年1 月1 日起，将全面停止对SHA-1 证书的支持。届时在最新版本的Windows 系统中，SHA-1 证书将不被信任。

而根据Chrome 官方博客的文章，使用SHA-1 证书且证书有效期在2016 年1月1号至2016 年12 月31号之间的站点会被给予「安全的，但存在漏洞」的提示，也就是地址栏的小锁不再是绿色的，并且会有一个黄色小三角。而使用SHA-1 证书且证书有效期超过2017 年 1月1号的站点会被给予「不安全」的红色警告，小锁上直接显示一个红色的叉。参考：[微软谷歌Mozilla 三大浏览器2016放弃对SHA-1证书支持](https://www.wosign.com/News/2015-1224-01.htm)

然而，并不是所有的终端都支持SHA-2 证书，服务端不支持还好办，浏览器只能依赖于用户升级了。下面是常见浏览器支持SHA-2 证书的最低版本：



可以看到，如果要照顾没有打XP SP3 补丁的IE6 用户，只能继续使用SHA-1 证书。

在我之前的文章中，还提到过ECC 证书，这种新型的证书支持度更差，这里略过不提，有兴趣的同学可以[点这里](https://imququ.com/post/optimize-tls-handshake.html#toc-2-1)查看。

是否可以针对不同浏览器启用不同证书呢?理论上服务端可以根据客户端Client Hello 中的 Cipher Suites 特征以及是否支持SNI 的特征来分配不同证书。现在有一些网站利用IE on Windows XP 不支持SNI 这个特性，配置使用SHA-1 证书的Default Server 给它们用。

本文先写这么多，很多策略都需要根据自己网站的用户来决定，例如我的博客基本没有IE8- 用户，理所当然可以禁用SSLv3。如果你的产品还有很多使用老旧浏览器的用户，那就必须为这些用户做兼容方案了。一种方案是：只把主域安全级别配低，将XP 上IE 用户的HTTPS 请求直接重定向到HTTP 版本，这样其它域名可以使用高安全级别的配置，运维起来比较方便。

**转载-如何正确使用HTTPS加密协议经验分享(三)**

发布日期：2016-05-17

关于「如何正确使用HTTPS加密协议经验分享」这个话题前面已经写过两篇文章：第一篇主要介绍[**HTTPS加密协议**](https://www.wosign.com/)如何与一些较新的安全规范配合使用，面向的是现代浏览器;第二篇主要讨论启用HTTPS过程中，在SSL版本、Cipher Suite、证书、SSL 扩展(如 SNI)等方面可能遇到的问题，以及在老旧浏览器下如何取舍。本文做为本系列最后一篇，将介绍一些启用HTTPS过程中的注意事项。

**资源替换**

HTTPS 网页中加载的HTTP 资源被称之为Mixed Content(混合内容)，之前的文章中详细介绍了Optionally-blockable 和Blockable 两类Mixed Content，也介绍了各种浏览器对Mixed Content 的加载策略。为了最好的用户体验，HTTPS 网站不要出现任何Mixed Content。换句话说，HTTPS 页面中所有资源都必须替换为HTTPS 的。

代码层的替换比较简单，但一些存在数据库中的文本(例如商品描述中的图片地址)就很容易遗漏，需要特别注意。

如果你的网站只打算支持HTTPS，将所有外链资源(CSS、JS、图片、音频、字体文件、异步接口等等)直接替换为HTTPS 地址，再把网站HTTP 请求重定向到HTTPS 即可。

当前很多支持HTTPS 的网站出于各种原因，针对老旧浏览器或特殊网络还是允许通过HTTP 访问。这时候，强烈建议让所有资源服务都同时支持HTTP/HTTPS 访问。这样只要页面在使用这些资源时省略协议部分，浏览器就能根据主页面协议类型来自动选择 HTTP/HTTPS 资源。例如：

<img src="https://example.com/static/img/blog/ququ.jpg" />

=>

<img src="//example.com/static/img/blog/ququ.jpg" />

针对现代浏览器，还可以通过upgrade-insecure-requests 这个CSP 指令，让浏览器自动替换。更多说明，[详见这里](https://imququ.com/post/sth-about-switch-to-https.html#toc-1-1)。

省略 URL 协议有个风险点：个别移动网络提供商篡改页面时，会将这种写法的 URL 改坏，导致资源无法访问。详见《[诡异问题排查之「DataURI 引发的血案」](https://imququ.com/post/datauri-and-404.html)》这篇文章。

**服务端代理**

在启用HTTPS 的实际过程中，本站的静态资源和接口相对容易改造，毕竟都可控。但很多第三方资源或接口就是不提供HTTPS，那就只能在服务端做一层HTTPS 代理。

服务端代理另外一个典型应用是用来解决跨域问题。通常代理本身要做的工作不多，直接用Nginx 做反向代理，或者用Lua、Node.js 等语言构建轻量中转服务都是不错的选择。但也有几点需要注意：

· 代理对请求Referrer、被代理的URL都需要做好白名单机制;

· 代理会造成第三方通过REMOTE\_ADDR 拿到的是代理IP，很可能导致这个IP 被限制请求频率或被封;

· 代理只能拿到自己域名下的Cookie，需要从其它域获取Cookie 的第三方接口被代理后可能不能正常工作;

另外，对于页面上通过iframe 嵌入的第三方HTTP 页面，如果要做HTTPS 代理，还需要修改页面里的所有资源链接，很容易出问题。对于这种情况，强烈建议联系第三方修改或者换产品方案，不要在HTTPS代理上耗费太多精力。

还有一个不那么常见的问题顺便说下：如果页面表单的action 地址使用了HTTP 地址，会导致Chrome 地址栏绿色小锁消失。

最后，再讨论一个第三方接口由本地服务提供的特殊场景(例如 Android APP 在本地开一个 127.0.0.1 的 HTTP 服务，给网页调用)。这个服务几乎不可能升级为 HTTPS，显然也无法使用服务端代理。对于这个问题，我在《[利用图片传输数据的另类思路](https://imququ.com/post/use-image-to-transfer-data.html)》这篇文章里，提供了一种能用但不完美的解决方案。

**Referrer**

目前大部分浏览器，在发生协议降级时默认不发送Referrer 信息，最典型的场景就是从HTTPS 页面点链接跳到HTTP 网站时，浏览器并不会在请求头中带上Referer 字段。对于给第三方导流的网站，这一点肯定无法接受。

针对现代浏览器，这个问题可以通过给页面加上下面这个meta 标签来解决：

<meta name="referrer" content="always" />

有关这个meta 的更多用法，请参考《[Referrer Policy 介绍](https://imququ.com/post/referrer-policy.html)》这篇文章。

针对老旧浏览器，这个问题可以通过在本站部署HTTP 跳转服务来解决，借助HTTP 页面把 Referrer 传给第三方。

另外，本文同时出现了Referrer 和Referer 两种写法，如果对此有困惑，推荐阅读《[Referrer还是Referer](https://imququ.com/post/referrer-or-referer.html)?》。

**连通性**

很多网站在启用 HTTPS 之后，都会接到无法访问的用户反馈。除去自身配置问题之外，很有可能是HTTPS 连通性受到了干扰。

最常见的干扰是运营商劫持了域名DNS 解析，这种劫持服务器一般会将用户请求反代到源网站，再在响应里夹带私货。在HTTP 时代，这种劫持多半只会造成页面出现广告，网站还能用;而升级到HTTPS 后，由于身份认证机制的存在，劫持服务器无法成功反代第三方网站，从而导致网站完全不可用。

我们最近在移动端做过一个统计：对比在HTTP 网站加载本域HTTP/HTTPS 空图片的失败率(我们对失败的定义是触发图片的error 事件，或者超过9s 仍未触发load 事件)，HTTPS 要高出三个百分点。

对于HTTPS 请求失败日志，还可以进一步分析，比如找出是哪些运营商 / 地域的 HTTPS 联通性比较差，从而有针对性做一些策略。由于每个业务情况都不相同，这里只抛出问题，详细的统计数据和应对措施不在这里讨论。

本文先写到这里，有新的注意事项我会随时补充进来。最后我想说的是，启用HTTPS 并不复杂，虽然会遇到各种各样的问题，但都能找到对应的解决方案，所需的无非就是决心、耐心和细心。在现代浏览器中，越来越多的新功能都限定在HTTPS 下才能使用，HTTPS 也是部署HTTP/2 的先决条件。HTTPS 和HTTP/2 已经成为WEB 服务的标配，赶紧行动起来吧。