最近大家在使用百度、谷歌或淘宝的时候，是不是注意浏览器左上角已经全部出现了一把绿色锁，这把锁表明该网站已经使用了 HTTPS 进行保护。仔细观察，会发现这些网站已经全站使用 HTTPS。同时，iOS 9 系统默认把所有的 http 请求都改为 HTTPS 请求。随着互联网的发展，现代互联网正在逐渐进入全站 HTTPS 时代。

因此有开发同学会问：

全站 HTTPS 能够带来怎样的优势？HTTPS 的原理又是什么？同时，阻碍 HTTPS 普及的困难是什么？

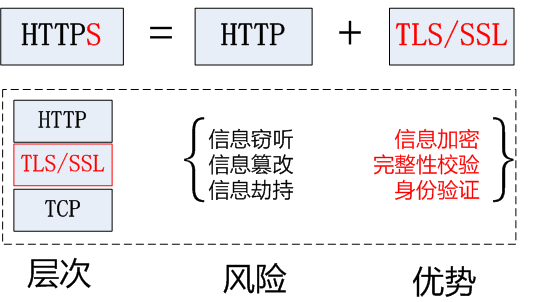
为了解答大家的困惑，腾讯TEG架构平台部静态加速组高级工程师刘强，为大家综合参考多种资料并经过实践验证，探究 HTTPS 的基础原理，分析基本的 HTTPS 通信过程，迎接全站 HTTPS 的来临。

**1.HTTPS 基础**

HTTPS(Secure Hypertext Transfer Protocol)安全超文本传输协议 它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)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。

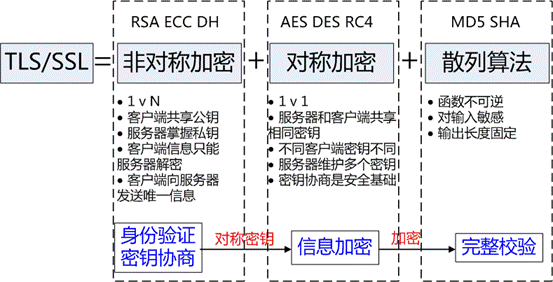
TLS/SSL 全称安全传输层协议 Transport Layer Security, 是介于 TCP 和 HTTP 之间的一层安全协议，不影响原有的 TCP 协议和 HTTP 协议，所以使用 HTTPS 基本上不需要对 HTTP 页面进行太多的改造。



**2.TLS/SSL 原理**

HTTPS 协议的主要功能基本都依赖于 TLS/SSL 协议，本节分析安全协议的实现原理。

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；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；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。

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。

**3.PKI 体系 3.1 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;

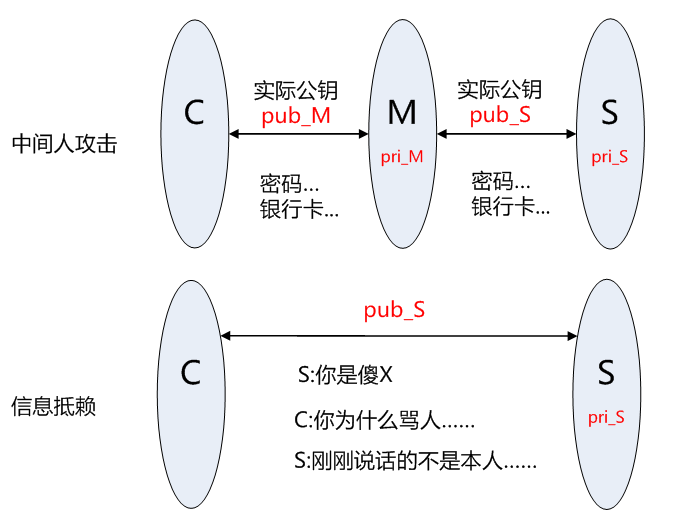
C 向 S 请求公钥时，M 把自己的公钥 pub\_M 发给了 C;

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。

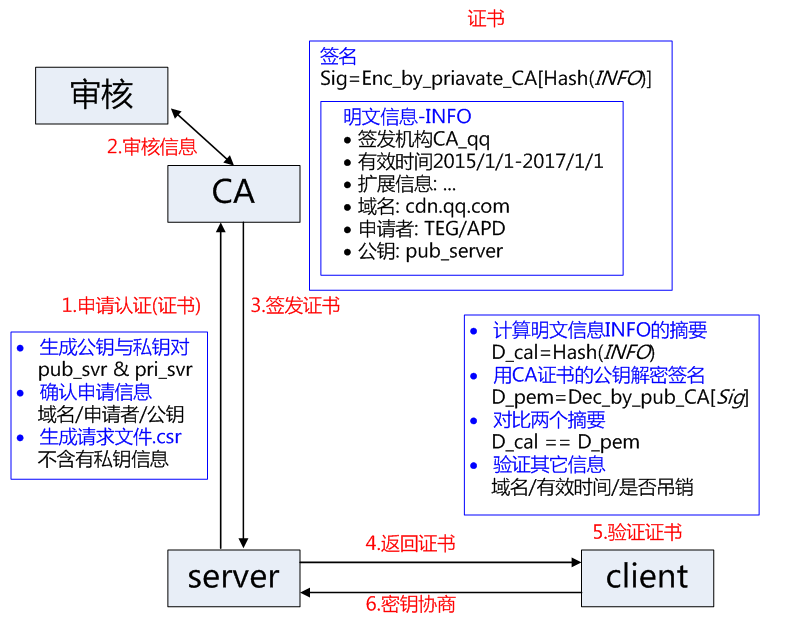
因此该方案下至少存在两类问题：中间人攻击和信息抵赖。



**3.2 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。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。

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；

d.证书=公钥+申请者与颁发者信息+签名；

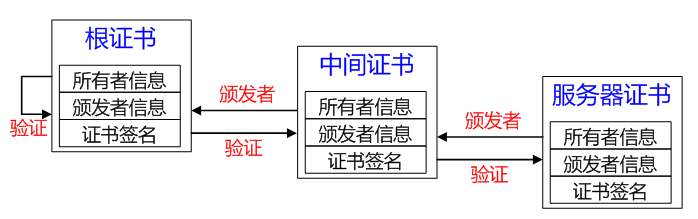
**3.3 证书链**

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，证书链的验证是自下而上的信任传递的过程。

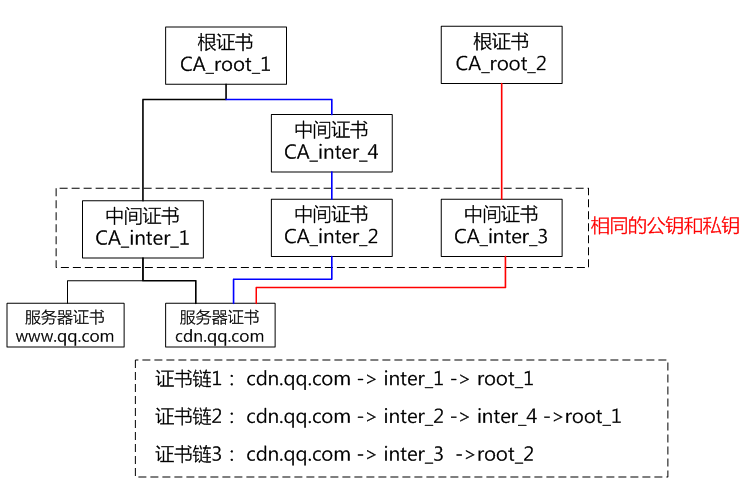
二级证书结构存在的优势：

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。

**3.4 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，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。

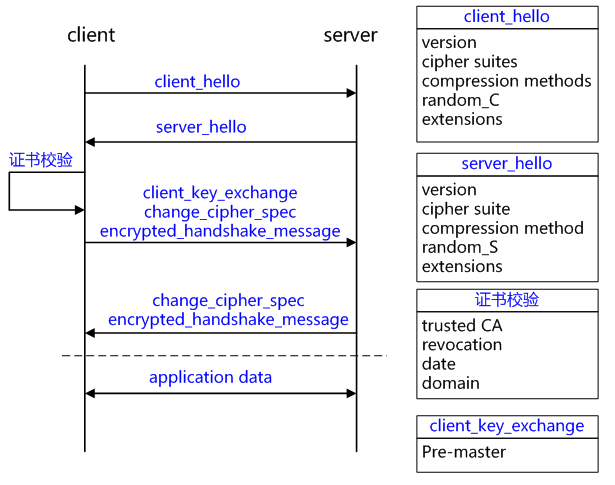
(b) OCSP

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。

**4.TLS/SSL握手过程**

**4.1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，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 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。

**4.2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。

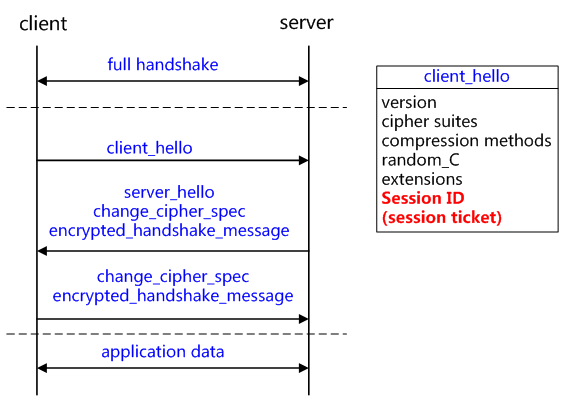
session ID 由服务器端支持，协议中的标准字段，因此基本所有服务器都支持，服务器端保存会话ID以及协商的通信信息，Nginx 中1M 内存约可以保存4000个 session ID 机器相关信息，占用服务器资源较多；

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；

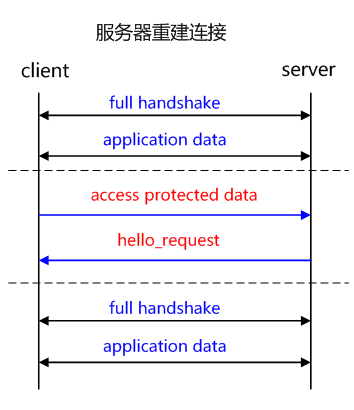
(d) 如果解密成功，则返回 change\_cipher\_spec 与 encrypted\_handshake\_message 信息，两个信息作用与 session ID 中类似；

(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。

**4.3 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**

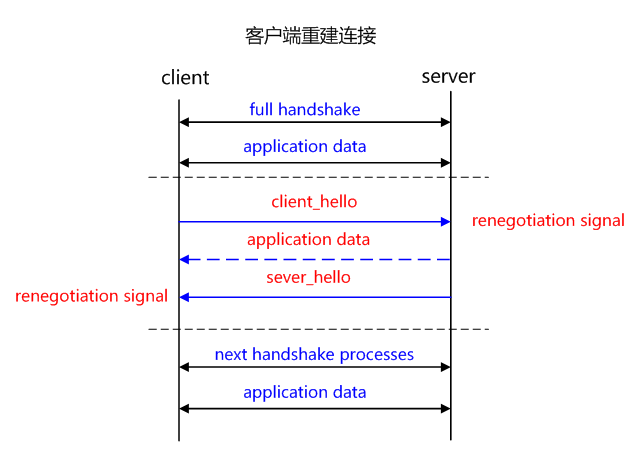
服务器端重建连接一般情况是客户端访问受保护的数据时发生。基本过程如下：

(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；

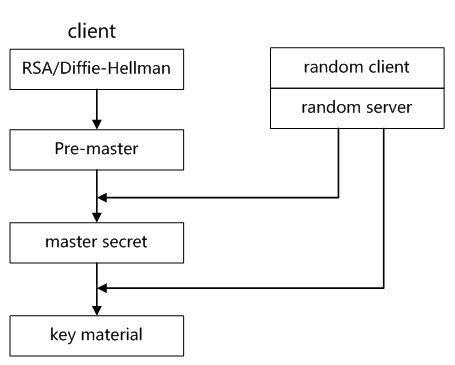
(g) 客户端和服务器开始新的重建连接的过程。

**4.4 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)

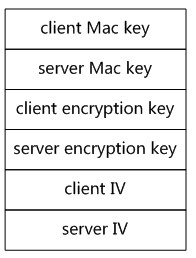
(b) 不管是客户端还是服务器，都需要随机数，这样生成的密钥才不会每次都一样。由于 SSL 协议中证书是静态的，因此十分有必要引入一种随机因素来保证协商出来的密钥的随机性。

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；

**4.4 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)。

**4.5 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。

**5.HTTPS 性能与优化**

**5.1 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。

**5.2 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。