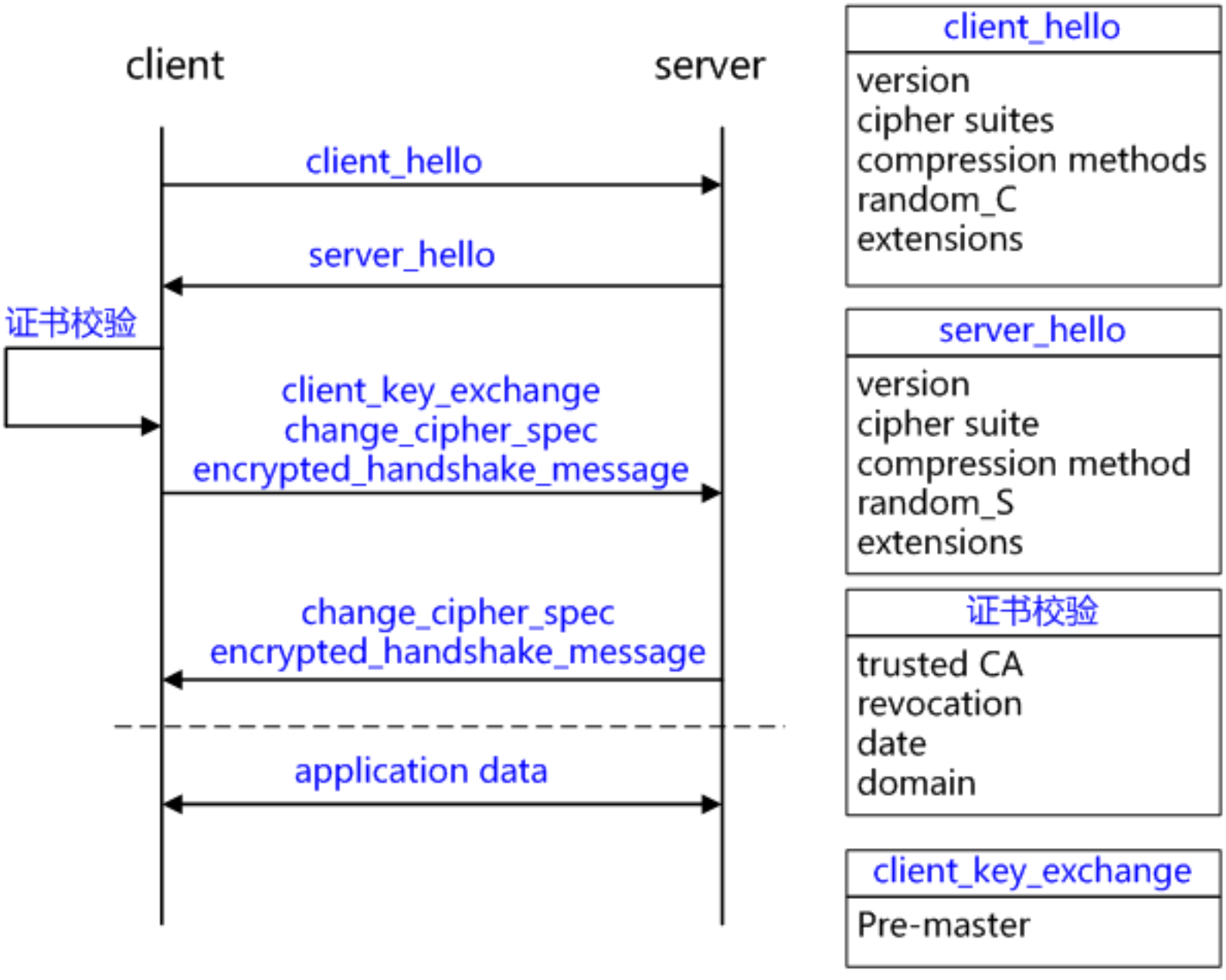
HTTPS协议详解(四)：TLS/SSL握手过程

本文大部分内容摘自：http://www.wosign.com/faq/faq2016-0309-04.htm 尊重知识产权，**转载注明Wosign**

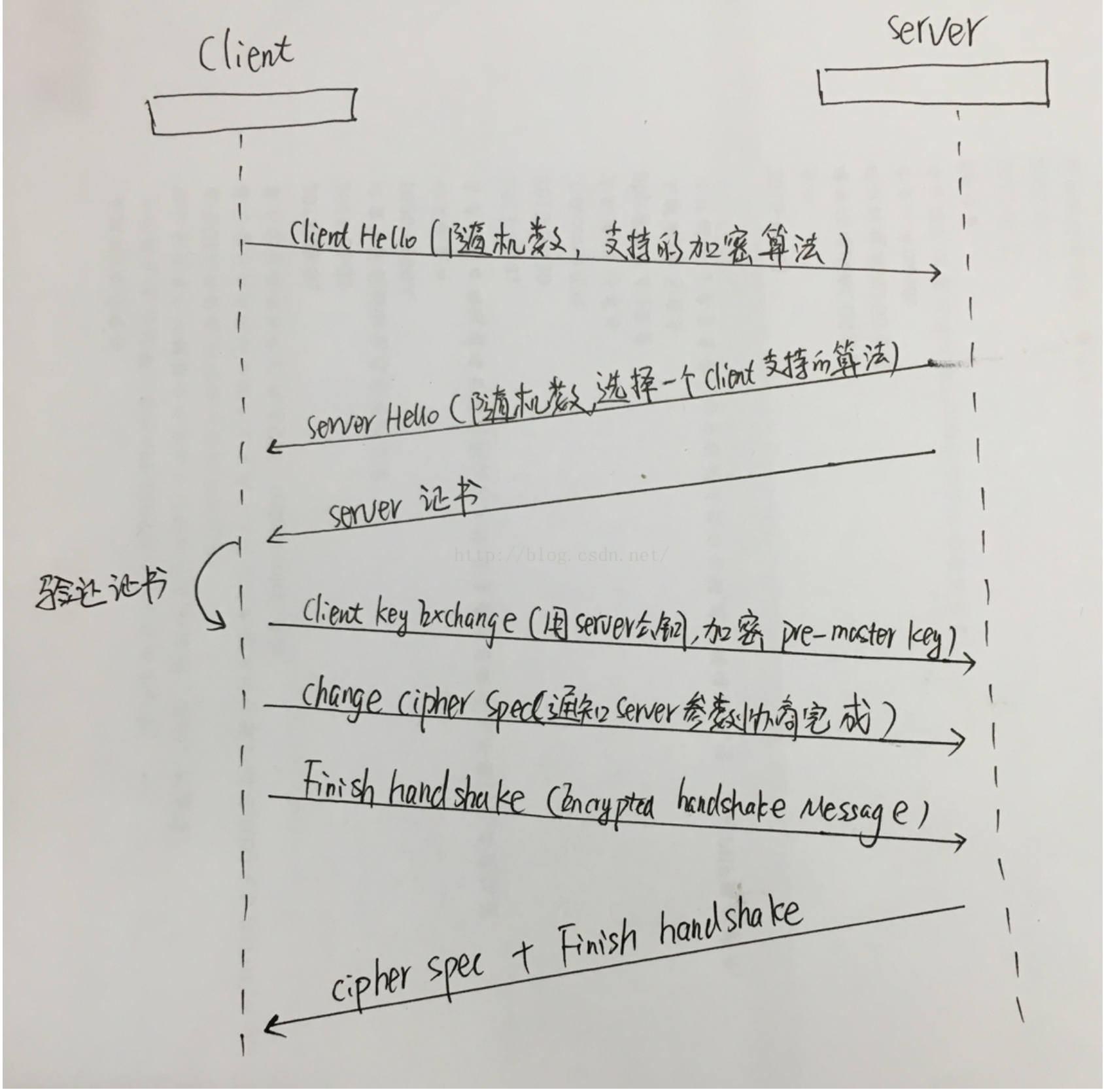
----------专栏导航----------

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


client
server
client_hello
server_hello
证书校验
client_key_exchange
change_cipher_spec
encrypted_handshake_message
change_cipher_spec
encrypted_handshake_message
application data
client_hello
version
cipher suites
compression methods
random_C
extensions
server_hello
version
cipher suite
compression method
random_S
extensions
证书校验
trusted CA
revocation
date
domain
client_key_exchange
Pre-master

再看一张手绘时序图



(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

• 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;

• 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;

• 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。

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。

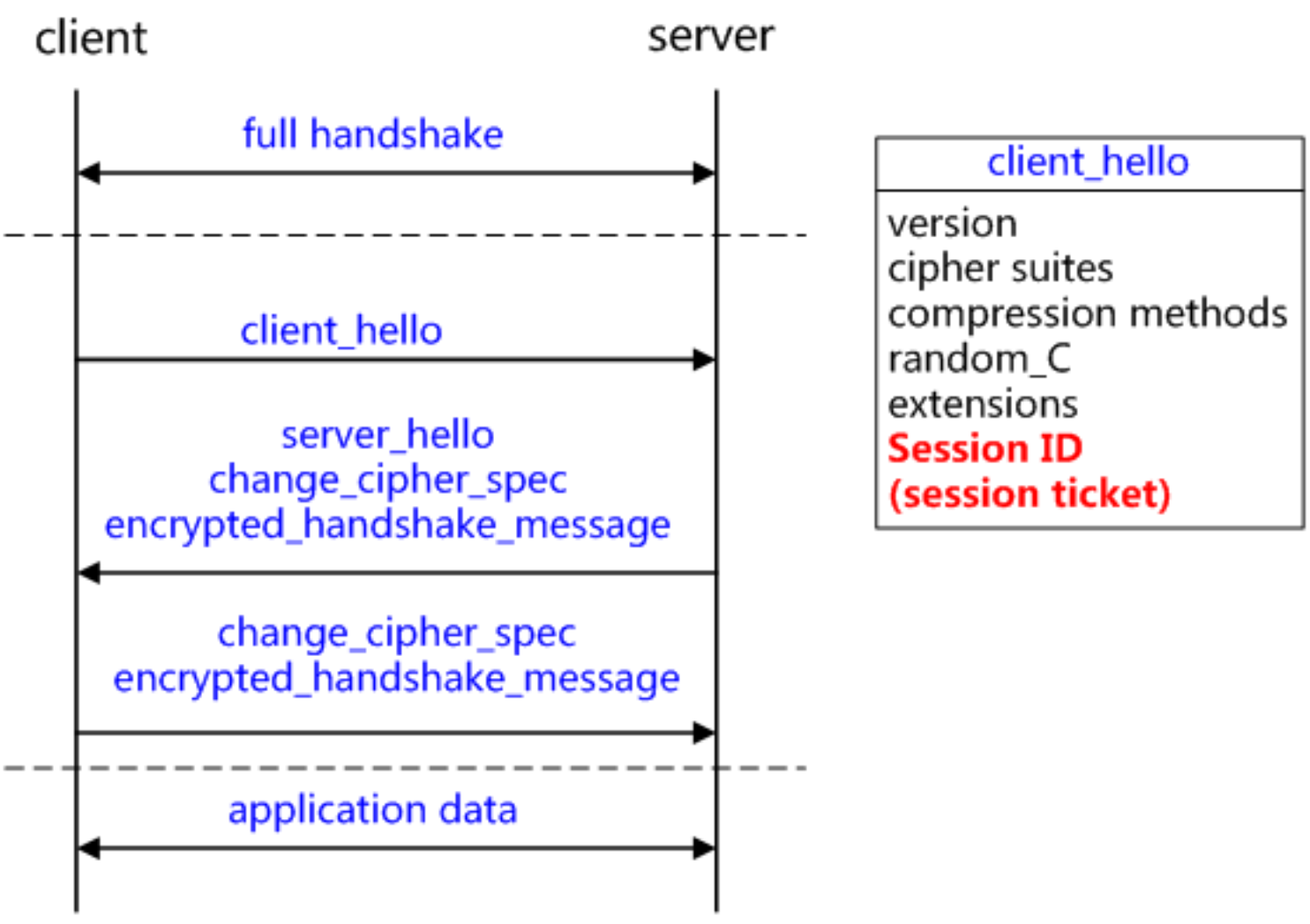
session ID 由服务器端支持，协议中的标准字段，因此基本所有服务器都支持，服务器端保存会话ID以及协商的通信信息，Nginx 中1M 内存约可以保存4000个 session ID 机器相关信息，占用服务器资源较多;

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;

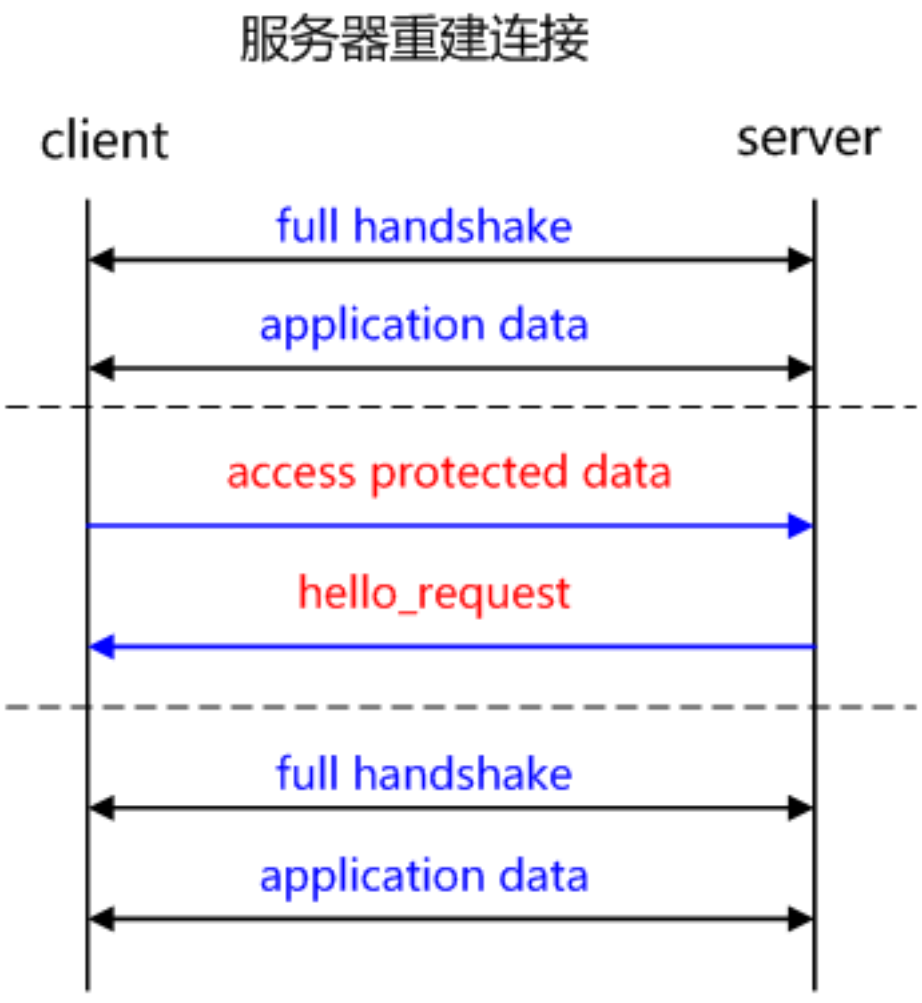
(f)如果客户端能够验证通过服务器加密数据，则客户端同样发送 change_cipher_spec与encrypted_handshake_message 信息;

(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：

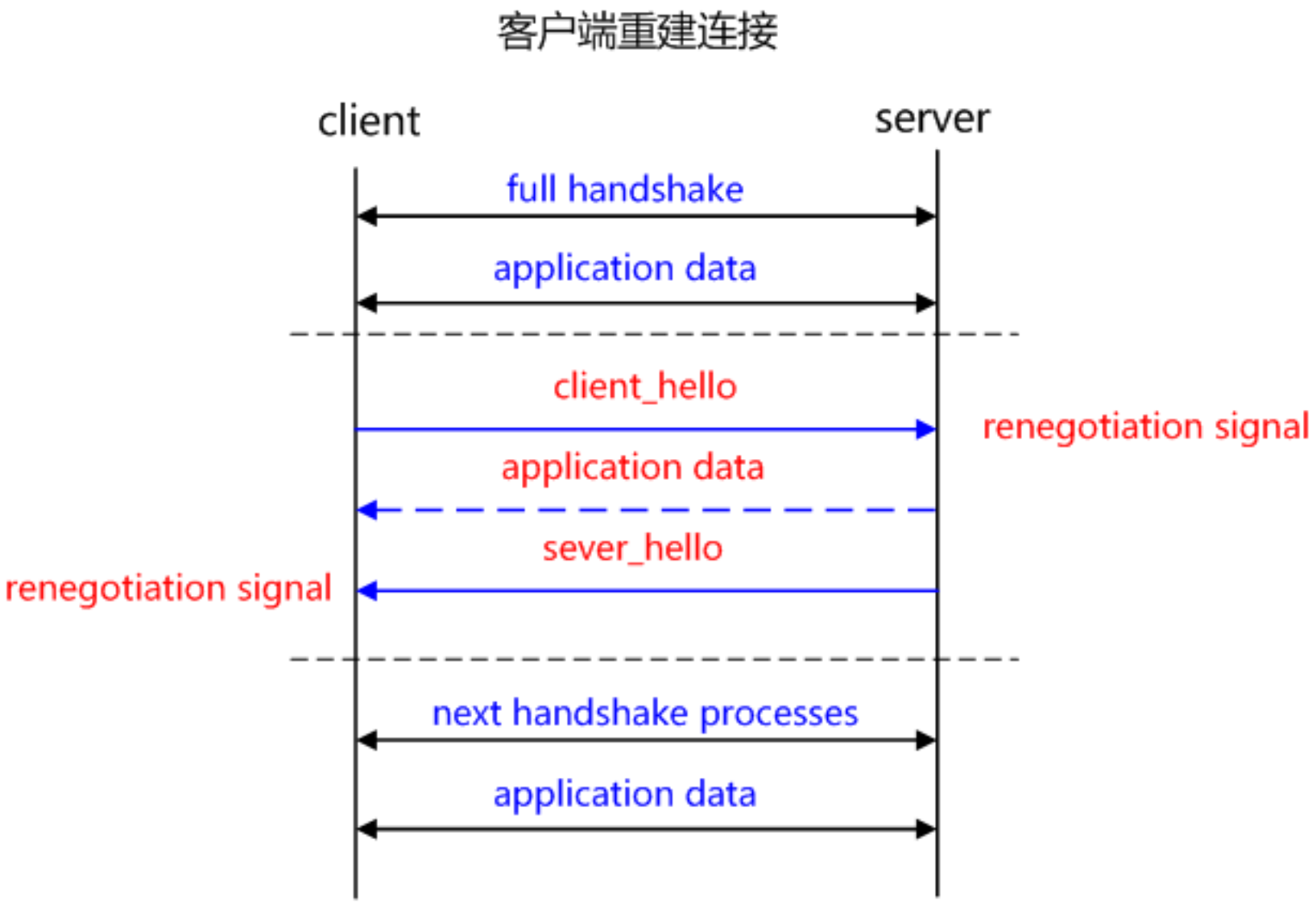
(a) 客户端和服务器之间建立了有效 TLS 连接并通信;

(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

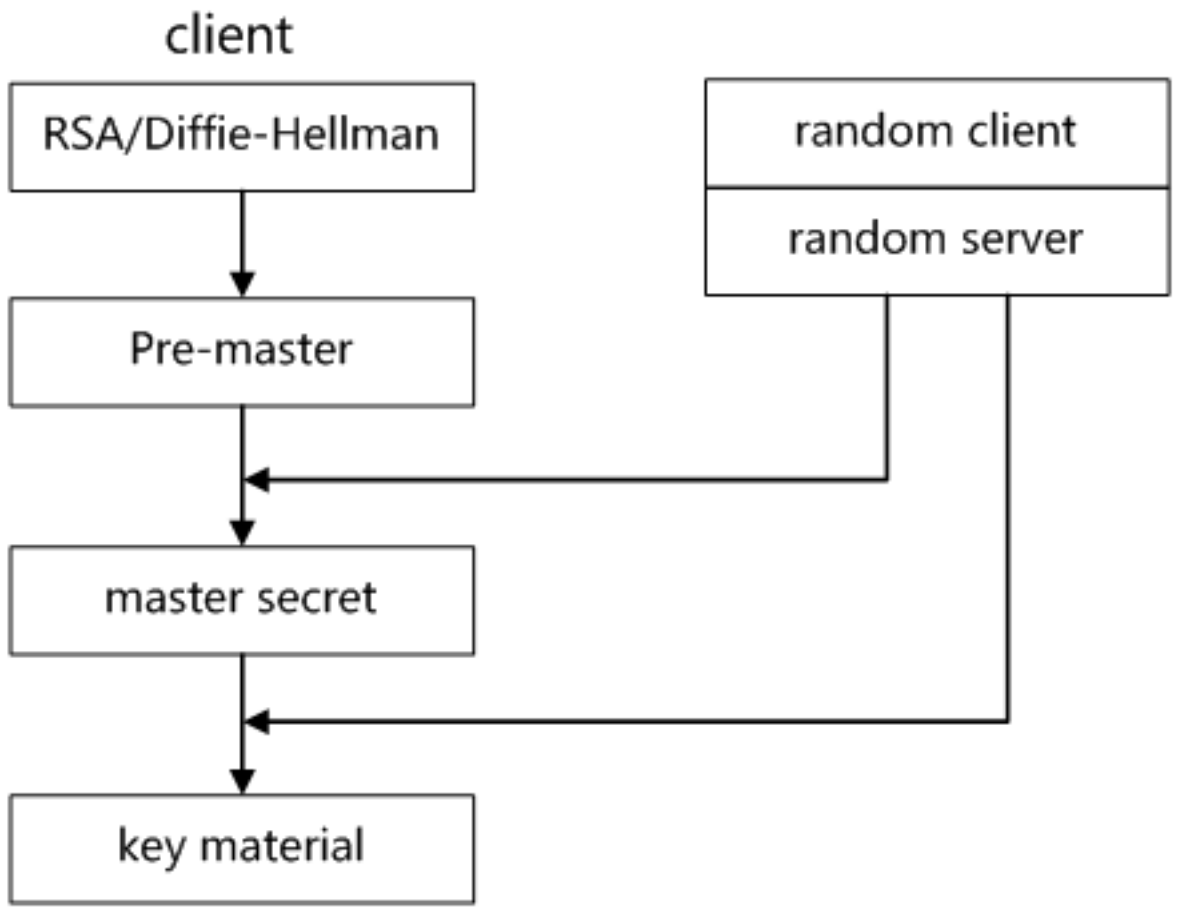
上节提到了两个明文传输的随机数 random_C 和 random_S 与通过加密在服务器和客户端之间交换的 Pre-master，三个参数作为密钥协商的基础。

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。

另一篇文章，容易理解一下：扫盲 HTTPS 和 SSL/TLS 协议[3]：密钥交换（密钥协商）算法及其原理

(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)

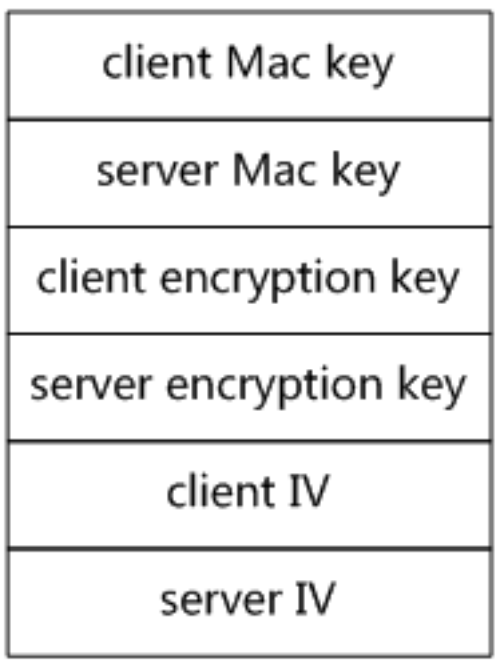
(b) 不管是客户端还是服务器，都需要随机数，这样生成的密钥才不会每次都一样。由于 SSL 协议中证书是静态的，因此十分有必要引入一种随机因素来保证协商出来的密钥的随机性。

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。