SSL 和 TSL
安全通信协议概要

Simon Horman aka Horms

horms@valinux.co.jp
horms@verge.net.au
horms@debian.org

翻译:Walter Liu

摘要

SSL/TLS 被广泛用于在因特网上安全的传输数据，但是它不是一个具有魔力的解决方案，如果没有很好的理解协议是如何工作的和底层的技术，就无法更全面的使用 SSL/TLS，甚至更糟的是，很容易以不安全的方式使用 SSL/TLS。

这次演示将会解释 SSL/TLS 是如何工作的，从数据完整性，保密性和端点验证这些高层协议的讨论到组成 SSL/TLS 协议不同报文和最终保护链接安全的加密技术这些低层讨论。

本次面向的观众是对使用或者开发利用 SSL/TLS 保护数据传输安全应用程序的人。

目录

1 安全通信

安全套接字层协议（SSL）和传输层安全协议（TLS）目的是提供一种在网络两端安全通信的机制，使得两端无需进行端到断的控制，也避免了第三方窃听通信内容。因特网就是很好的例子。本质上，我们将会谈到连个方面，数据完整性和端点验证。

1.1 数据完整性

当两方以安全的模式通信时, 重要的是数据要完整地, 未加修改的接受, 并且没有第三方探测或者修改通信数据。为了保证数据完整性,SSL/TLS 使用了多种加密方法。非对称加密和对称加密能阻止第三版访问报文内容，即使报文被窃取，从而提供隐私。也能防止报文被删除或者插入。报文摘要可以防止报文被修改。

1.1.1 非对称加密

非对称加密，通常也称作公钥加密，允许通信双方进行加密通信，而无需事先商定密钥。如果加密和揭密使用不同的密钥, 那么就称作非对称加密。无可辩驳，最著名的非对称加密算法就是 RSA。

RSA 密钥由两个部分组成，公钥和私钥。望文生义，公钥就是任何感兴趣的一方都可以自由获取的，而私钥是保密的。私钥用来对报文加密，公钥则用来验证。只有私钥的拥有者可以加密报文，任何可以访问公钥的人都可以验证报文。相反地，公钥可以加密报文，只有私钥可以解密报文。

RSA 算法的强大之处在于，两个大质数的乘积很难被因式分解。质数越大，密钥越强劲。质数的长度和产生的密钥导致操作处理非常慢。其他的非对称加密算法都是差不多慢，所以对于大块的数据传输不适用。因此，SSL/TLS 使用非对称加密验证，事先商定对称加密密钥传输大块数据。在这点上，非对称密钥的优点在于无需再通信之前商定密钥，而且某种程度上也减轻了非对称加密

tigat 的慢速。

1.1.2 对称加密

对称加密允许双方共享一个密钥来进行加密通信。之所以被成为对称加密，是因为加密和解密用的是同一个密钥。常见的对称加密算法有 DES,DES3 和最近的 AES。

因为加密和解密使用的是同一个密钥，所以双方以安全的方式共享这个密钥就非常重要。如果第三方获得了密钥，就可以伪造加密报文，也可以解密已加密的报文。如果端点双方都在同一个私有网络，比如和银行通信的 ATM 和 POS 机，这通常不是什么大问题，只要密钥被事先放在设备里。如果对于更常见的临时通信，正如 SSL/TLS 提供的，相互信任关系需要在运行中确定，因此用于对称加密的密钥是哟你非对称通信来获得。对称加密算法倾向于使用比非对称加密更短的密钥，这通常可以提供相似程度的安全。这导致再软件中加密算法非常快，这非常重要，因为现在大多数的 SSL/TLS 都是再软件中实现的。

1.1.3 分块

对称加密算法是基于块的。也就是加密一定量的数据，然后解密。但是 SSL/TLS 提供基于流的数据传输，通常数据的数量和加密算法使用的块不想匹配。再任何情况下，传输的数据和使用的算法是独立的，所以需要分块，把数据分解为块，如果必要的话，填充块。

一个简单的分块实现是仅仅把数据分解为块大小的块，如果必要则填充块，然后加密得到的块。这种方法叫做**电子密码本**（**ECB**) 模式。它的最大的缺点就是如果两块文本是相同的，那么密文也将相同。这个信息对于试图解密流的攻击者是非常有利的。

因此，SSL/TLS 使用**密码分组链接（CBC）**模式。当传送数据流时，第一块文本在加密前和一个***初始向量*（IV）**异或。再 SSL/TLS 中，初始向量 (IV) 再握手商定密钥时产生，并且每次连接都不一。样。后一块文本和前一块密文再加密前异或。这样第一块后的每一块都依赖于前一块。所以尽管流里的两块文本相同，密文也不同。

1.1.4 报文验证码

报文验证码（**MAC**）保证报文在传送过程中没有被修改和丢失。它可以被看作是包含密钥的报文摘要。在数据发送时构造，在数据接收时验证。如果不知道输入文本和密钥，是不可能伪造摘要的，因此，攻击者需要知道密钥才能为一个改变的报文重新构造一个有效的 **MAC**。为了防止再次攻击，SSL 和 TLS 在构造 **MAC** 时包含了单调递增的数字。

1.2 端点确认

在双方通信时，确认对方确实是声称的对方也是非常重要的。在 SSL/TLS 中，这是通过证书来达到的。在建立 SSL/TLS 连接过程中，一个使用端点证书签字的报文和证书一起发送。证书本身是由认证机构签发的，网络信任是建立在认证机构。

SSL/TLS 是客户端 -服务器协议。这和高层的客户端 -服务器协议比如 HTTPS, POP3S 和 IMAPS 很类似。这种情况下，典型地是客户端验证确认服务器。也就是，网络服务器或者邮件服务器拥有证书，而客户端没有。但是，SSL 和 TLS 允许服务器对客户端进行确认，这可以用来控制服务访问。匿名密钥交换也是可以的，在这种情况下，服务器不会发送自己的证书。这对攻击这伪造的服务器没有任何安全保证。

为了验证证书的合法性，需要做多次检查。基本的检查包括证书没有过期，检查它的名称和主机名相同。证书也需要加密确认，以确保是由某个认证机构签发的。提供认证机构列表来检查是由软件负责的。以 HTTPS 为例，网页浏览器通常包含了认证机构的列表，也可以在安装后添加或删除。服务器使用自己签发的证书也是很常见的，在这种情况下，通常由访问该网站的个人来确认证书的合法性。比如，检查离线足迹。尽管通常希望它是合法的。

2 版本

SSL 的第一个发布版本是 SSLv2, 由 Netscape 发布。但是他有一些缺陷。

- **MAC** 比较弱，依赖于 **MD5**
- 验证和加密使用相同的密钥。使得出口密码 MAC 非常弱
- 使用 TCP 关闭数据链接，这意味着如果伪造 TCP FIN[1]报文，可以导致截断攻击。
- 在攻击中，无法手动对握手进行保护

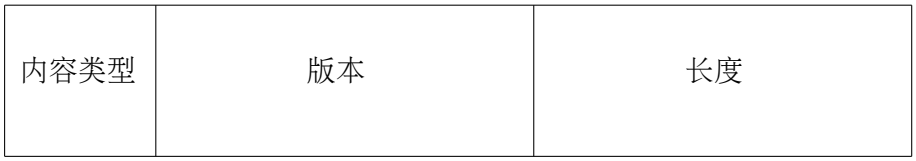
这写问题被 SSLv3 解决，SSLv3 也是由 Netscape 发布的。随后被 **IETF** 采用，并标准化为 TLSv1。SSLv2 和 TLSv1 主要的区别在于:

- 从主密钥提高密钥扩展，主密钥在握手期间通过数据交换计算得到。
- SSLv3 基于 **HMAC** 早期修改版。TLSv1 使用 **HMAC**。
- 实现必须支持 **DH/DSS** 密钥交换和 **3DES**。

[1]原文为 SYN

3 记录层

记录层封装了即将传送到底层协议 (通常是 TCP/IP) 的报文。每个记录可以包含 2^{14} 字节的数据，为了满足这个大小限制可以对报文进行分片。一个记录也可以包含多条报文，只要它们是同意类型。这通常发生在握手阶段，为了把多条报文放在同一个数据包里传输，提高握手速度，一条记录通常包含多个报文。

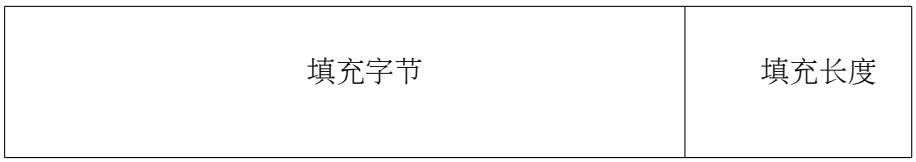


记录头

记录从记录头开始，包含了协议版本号，数据长度（以字节为单位）和报文类型，包括 change cipher spec, 警告 (alert), 握手（handshake) 和应用程序数据 (application data)。

紧接着记录头的是报文数据。数据是用商定的压缩算法压缩过的。然后，由于 TSL 和 SSL 中，并没有明确规定使用何种压缩算法，通常这个操作被省略。

然后对压缩的数据计算 **MAC**，并附加到记录之后。**MAC** 的计算方法在下一部分涉及。



记录填充

如果块密码对于连接生效，那么就会填充报文，使得报文大小是密码块大小的倍数。填充字段包含填充数据和一个字节的填充长度。填充长度（一个字节）所在字段也被包含在填充长度里。填充长度最大可以达到 255 字节，只要保证最终记录长度是密码块长度的倍数。

举个例子，如果记录为 123 字节，块长度为 8 字节，那么填充四字节后，记录，填充加上填充长度本身就会有 128 字节，是块长度的整数倍。12,20，和 252 也是合法的填充长度。允许填充一系列的填充值某种程度上使得记录长度模糊不清。

4	4	4	4	4
填充字节 1	填充字节 2	填充字节 3	填充字节 4	填充长度

记录填充

如果密码为 **NULL**, 或者是流密码，那么填充就不是必须的。实际中，这通常发生握手是构建在文本基础上，并且使用 **RC4**。

3.1 报文验证码

TLS 使用的报文验证码（**MAC**）是 **HMAC**,SSLv3 使用的是 **HMAC** 的早期草稿版，因此称作 **SSL3-MAC**。

HMAC 可用下面的等式表达:

$$H(K \oplus \texttt{opad}, H(K \oplus \texttt{ipad}, \texttt{text}))$$

其中:

$\oplus$: 是连接
text 是要加密的明文
H: 是哈希函数
B: 是以字节为单位的 H 长度
K: 最多 B 字节长的密钥
ipad: B 个字节 0x36
opad: B 个字节 0x5C

哈希函数是一个黑盒，因此可以使用大量不同的报文摘要算法。但是，所以非空密码族使用 **MD5** 或者 **SHA**。

在握手商谈密钥阶段，就计算将使用的密钥，并且是私密的。这非常重要，因为只要密钥是私密的，就能保证 **MAC** 的完整性，尽管底层哈希算法相对比较弱。简而言之，这是因为要攻击哈希函数，必须得到密钥的全排列。这对于 **MD5** 和 **SHA** 尤其重要，因为这两种算法都是有缺陷

的。这意味着冲突甚至比生日遵循的 $2^{B/2}$ 还要高，当报文摘要相同时，就会发生冲突。

SSL3-MAC 和 **HMAC** 拥有同样的属性，因为它就是 **HMAC** 的早期版本，只是在填充字符串处不同，ipad 和 opad 被追加在密钥之后，而不是和密钥异或。但是还没有像 **MAC** 那样受到更多密码学意义上的分析。

SSL 和 **TLS** 除了包含密钥，还包含用来计算 **MAC** 的递增序列号，这用来防止重复攻击。对于 **SSL** 连接, 每个发送出去的记录都有一个独立的序列号。序列号随着发送端和接受端发送数据和接受数据而独立的更新。

MAC 应用与记录的头部和数据。在压缩过数据以后（尽管没有明确的压缩算法，压缩过程通常省略），**MAC** 和记录的余下部分一起加密。

4 报文

在 **SSL** 和 **TLS** 中，报文是最基本的通信单元。有四种不同类型的报文。握手，change cipher suite 和警告都是控制报文，应用程序数据报文在连接的两端之间传送数据。

4.1 握手

握手报文头

握手报文头由类型和报文长度构成。类型包括：**hello** 请求，客户端 **hello** , 服务器 **hello**, 认证，服务器密钥交换，认证请求，服务器 **hello** 完成，证书确认，客户密钥交换，完成。用来上顶连接参数的握手报文的使用在连接请求部分涉及到。

4.1.1 Hello 请求

这个报文的主体是空的。服务器可以在任何时候，请求客户端开始握手商谈序列。如果商谈正在进行，或者由于某种原因不想商谈会话，客户端可以忽略这个报文。如果客户端不想商谈，它可以选择发送无需再商谈的警告。

这个报文没有包含在报文散列里，其他所有的握手报文会更新报文散列，并且报文散列包含在完成握手报文里。

在客户端初始化 **SSL** 或 **TLS** 连接时，这个报文不是必须的，因为握手总是初始化了的。但这个报文允许服务器请求会话重商定。典型的是，为长期连接重新商定密钥。

如果客户端希望重商定，它发送一个客户端 hello 报文，像 hello 请求一样，客户端 hello 报文可能被服务器忽略，也可能服务器回复不再重商定警告。

4.1.2 客户端 hello

当客户端连接到服务器初始化握手时，发送客户端 hello 报文。在握手头部后的是客户端版本号，指明了客户端支持的最高协议版本。**SSL** 和 **TLS** 实现需要支持以往的任何一个版本。接着是随机数，包含了当前 Unix **GMT** 时间戳和由密码安全伪数生成器产生的 28 字节数字。这些值在密钥材料生成中使用。**SSL** 和 **TLS** 没有要求客户端和服务器的时钟保持同步。

在随机数后面的是会话 ID。如果提供了会话 ID，通过重复使用之前服务器和客户端商定的密钥材料，可以简化握手序列。

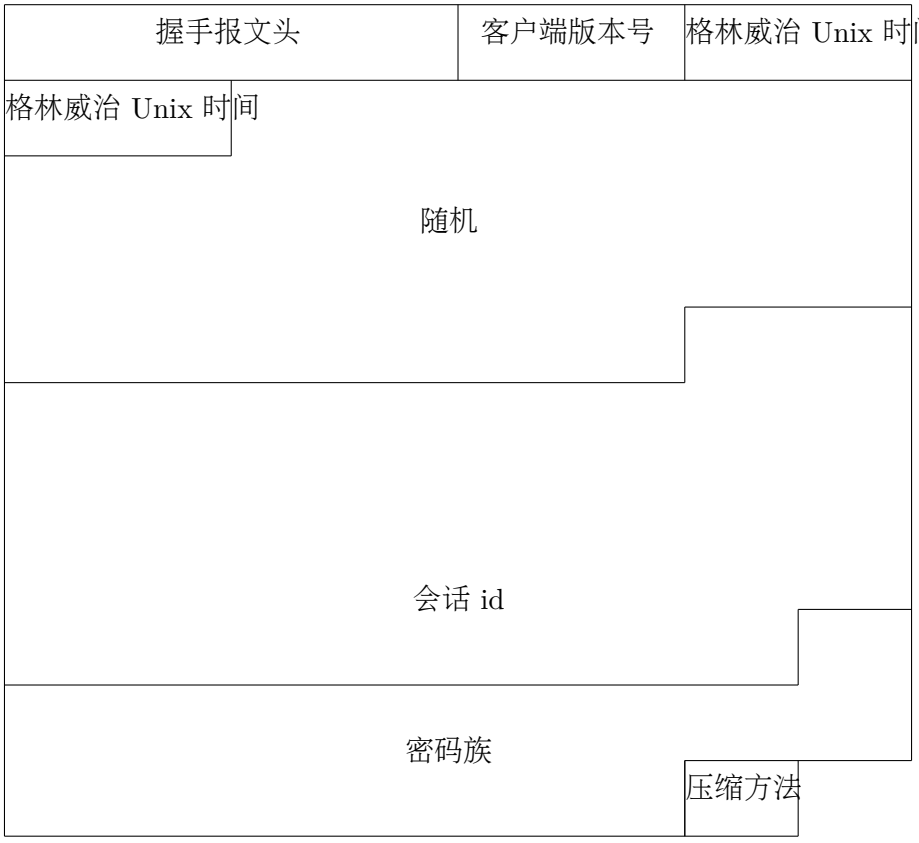
接下来的时一系列客户端准备使用的密码族。密码族定义了加密和散列函数，在握手结束后使用，也在握手阶段密钥交换时使用。用 8 比特数字表示密码族，在 **SSL** 和 **TLS** 说明，或其他附加说明中有相关文档。比如 **TLS_RSA_WITH_RC4_128_SHA** 是一个密码族，使用 **RSA** 作为密钥交换算法，**RC4** 作为数据加密算法，**SHA1** 作为 **MAC** 的散列函数，并表示成 0x00,0x05。不同的密码族提供不同级别的安全，因此使用的密码族对链接的安全程度起非常大的作用。

服务器可以随意选取密码族，并且如果服务器不支持任何一种密码族，它就会发送握手失败警告，然后关闭连接。尽管实现通常按照偏爱程度排序，因为服务器是任意选取，然而还是建议以用户感觉满意的安全级别列举。

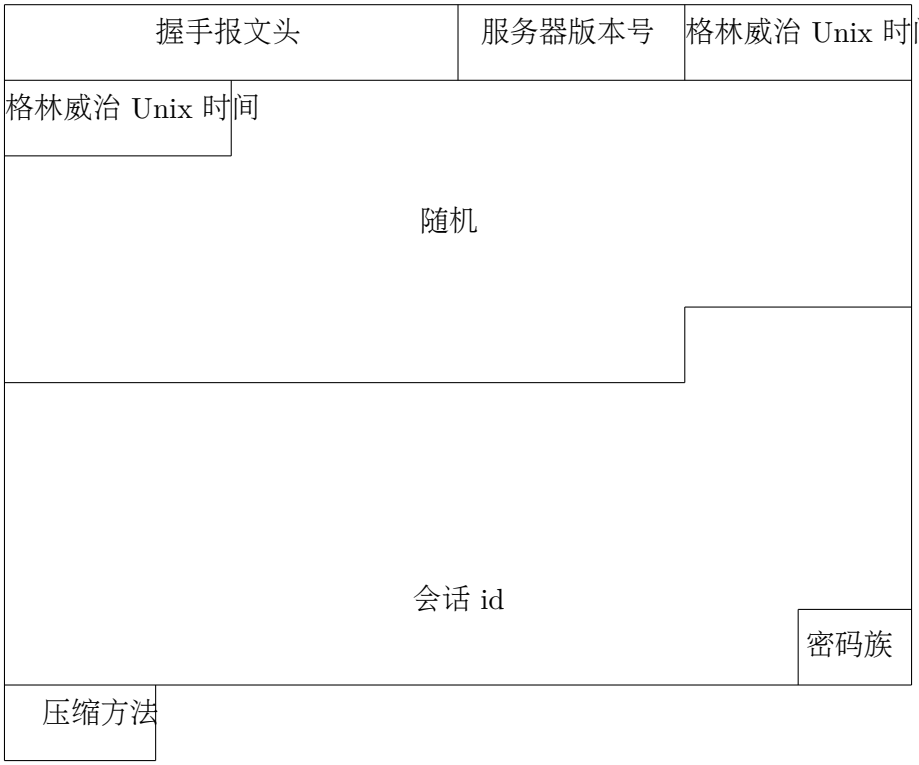
压缩方法用单个 8 比特数字表示，像密码族一样，客户端列举所有的压缩方法。然后，定义的唯一一个压缩方法是 **NULL**,0x00, 这通常是客户端提供的唯一压缩方法。

4.1.3 服务器 Hello

服务器 hello 报文是服务器发送响应客户端 hello 报文的。如果客户端版本被服务器支持，那么服务器版本号等于客户端版本号，否则就是服务器端支持的最高版本。接下来是随机数，包含了当前 Unix **GMT** 时间戳和由密码安全伪数生成器产生的 28 字节数字。



客户 hello 报文



服务器 hello 报文

服务器证书和客户端证书报文

后面的是可选择的会话 ID. 如果会话 ID 和客户端发送的会话 ID 相同，那么握手就通过使用被客户端和服务器缓存的密钥材料实施。如果为空，则说明服务器不愿意进行简化的密钥握手。这个 ID 就是将来客户端用来请求进行简化握手的 ID，客户端可以忽略它。服务器也可以选择忽略客户端发送的会话 ID，例如因为密钥材料没有被缓存。

最后，报文包含密码族和压缩方法，从客户端 hello 报文提供的列表里选择。

4.1.4 服务器认证

服务器在发送完服务器 hello 报文后，如果密码族指明是非匿名密钥交换方法，典型的做法就是发送服务器证书报文。它是由 **ASN.1** 编码序列的 **X.509v3** 证书组成，以服务器的证书开始。

4.1.5 服务器密钥交换

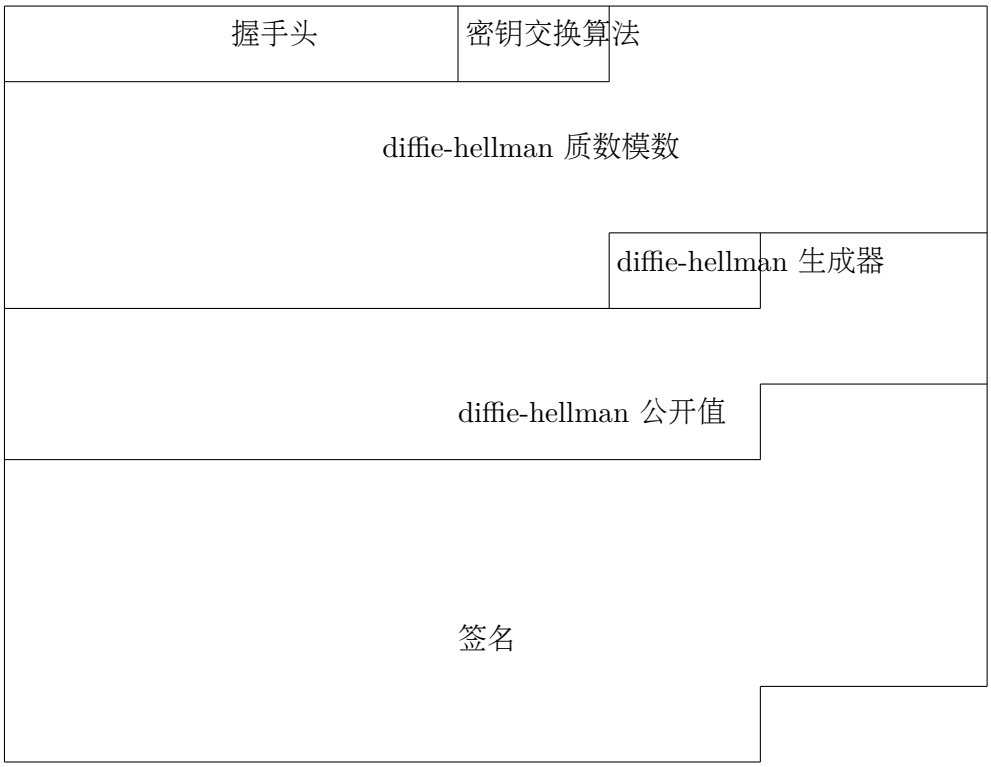
如果服务器证书报文没有包含足够的信息让客户端交换密钥，那么就发送服务器密钥交换报文，或如果使用匿名密钥，则在服务器 hello 报文之后发送。更确切的说，它是用来为匿名 **Diffie-Hellman**,**Ephemeral Diffie-Hellman** 和 **Ephemeral RSA** 密码交换方法服务。

Ephemeral 和匿名 **Diffie-Hellman** 是由选定的密码族决定的。当使用 RSA 出口密码族，并且公钥长于 512 比特时，使用 **Ephemeral RSA**。出口密码族是为了满足美国宽松出口限制的要求，在其他方面限制 RSA 密钥在 512 比特之内。**Ephemeral RSA** 允许服务器拥有一个强健的密钥和非出口密码族进行密钥交换，小于等于 512 比特的临时密钥通常在出口密码族中使用。它也允许服务器拥有一个不易受攻击的证书。此证书既用于出口也用于非出口密码族。

这个报文有两种格式，一个对应于 RSA，另一个对应于 Diffie-Hellman。两种都是在握手头后跟一个密码交换算法 –RSA 或 Diffie-Hellman。

报文的 RSA 版本包括 RSA 模数和指数。签名依赖于服务器证书和密钥交换的类型。对于

RSA 服务器密钥交换报文



Diff-Hellman 服务器交换报文

RSA，签名由 MD5 和 SHA1 的散列连接而成，由服务器密钥签证。对于 DSS，由服务器密钥对 SHA1 散列签证而成。散列的输入是客户端随机数，服务器随机数和报文从握手初到末的签名。

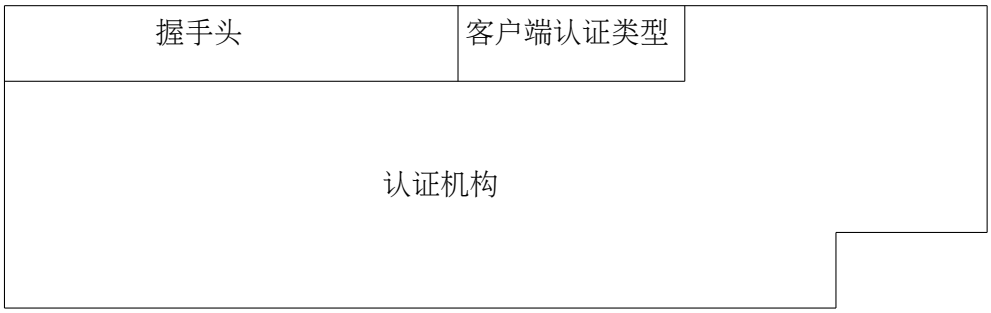
如果进行的是匿名密钥交换，签名就被忽略。必须指出的是，在这种情况下，没有办法验证服务器的身份，因此伪造服务器建立攻击是很容易的。

Diffiea-Hellman 版本的报文包含 Diffie-Hellman 指数模数，生成器和公开值。接下来的就是签名。

4.1.6 证书请求

这个报文由服务器发送请求认证客户端。报文由握手头开始，接着是一系列可接受的证书类型。已定义的阿证书类型由 rsa sign, dsa sign, rsa fixed dh 和 dss fixed dhc。最后，是一系列可接受的签证机构, 以 X.509 格式。这个列表不是用于 Diffie-Hellman 证书，因为验证在于客户端生成同样 pre-master 密钥。

这个报文在服务器密钥交换报文发出后发送，否则在服务器证书报文之后。如果使用匿名密钥交换，发送这个报文就是不合法的。



RSA 客户端密钥交换信息

4.1.7 服务器 Hello 完成

这个报文由服务器发送，表明它已经发送了 hello 报文和相关的报文。也就是说，它不再发送下面的报文：服务器 hello, 服务器证书，证书请求。这个报文除了包含握手头外，没有其他任何字段。

4.1.8 客户端证书

这个报文在接受到服务器 hello 完成报文后，立即发送，前提是受到了证书请求报文。像服务器证书一样，它包含了由客户端证书开头的 ASN.1 编码序列的 X.509v3 证书。

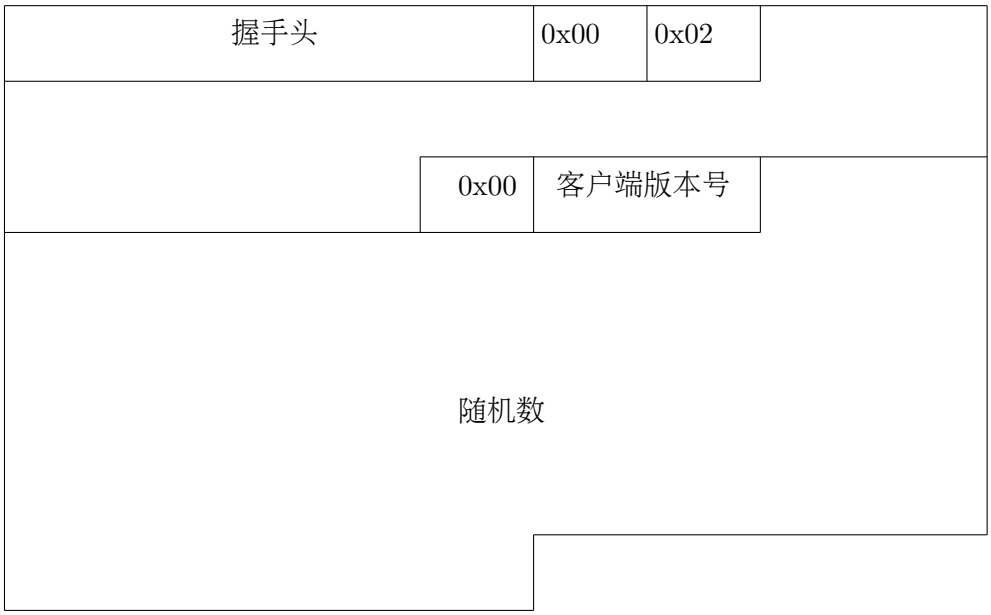
4.1.9 客户端密钥交换

客户端证书报文发送后，发送客户端密钥交换报文，否则在收到服务器 hello 完成报文后发送。它有两个变种，一个是 RSA 作为密钥交换算法, 另一个是 Diffie-Hello。

当使用 RSA 时，客户端密钥交换报文由握手报文头加上 PKCS# 编码的 pre-master 密语，密钥使用服务器密钥进行加密，服务器密钥包含在服务器证书报文里。握手报文头没有加密。

pre-master 密语由客户端支持的最大版本号加上 48 字节的随机数，随机数由密码随机数生成器产生。这 48 字节为主密语提供了客户端输入，会话密钥会继承自主密语。包含版本好的原因是组织在攻击时版本回滚。这个在 TLS 中确实有效，但是 SSLv3 使用商定版本号，因此检查这个值导致了一些不兼容问题。

PKCS#1 编码是必需的，因为要填充 pre-master 密语达到 RSA 密钥长度。它由 0x00,0x02, 随即生成填充字节，0x00 和报文，这种情况下是 pre-master 密语构成。填充字符的数目使得最终的报文长度和密钥相同。填充字节最少是 8, 所以最小的填充长度是 11 字节。理论上，报文分片可能是必要的，但是在实际中 RSA 密钥长于 59 字节 —pre-master 密语加上最小填充字节长度 –所以



RSA 客户端密钥交换报文

通常不分片。

因为报文是用服务器端的公钥加密的，所以只有私钥的拥有者可以解密报文。这就意味着，尽管证书可能由任意一方发送，只有拥有私钥的服务器才能成功的完成握手。这也保护了版本号不受攻击的侵犯，因为随机字节不可能从从修改的报文中恢复，并且如果他们不匹配，密钥材料就不匹配，当完成报文被处理时，握手将会失败。然后，正如上面讨论的，这种保护对于 SSLv3 是无效的。

当使用 Diffie-Hellman, 客户端密钥交换报文有握手头和 Diffie-Hellman 公共值构成。这允许客

Diffie-Hellman 客户端密钥交换报文

证书验证

户端和服务器计算同样的 pre-master 密语。

在使用客户端认证的情况下，客户端证书中包含使用服务器制定参数的 Diffie-Hellman 公共值，公共值在这个报文中被忽略，因为它已经知道了。在这种情况下报文头构成了整个的报文。

4.1.10 证书验证

客户端验证报文用来表明客户端确实是客户端证书的拥有者，它在客户端证书信息发出之后，立即发送。它是由所有已发送和接受握手信息的签名摘要构成。只有证书所有者的似有密钥可以对信息签名。只有这样才存在通过客户端证书验证客户端。

对于 TLSv1, 签名的格式和服务器密钥交换报文相似。对于 RSA 密钥，是摘要的 MD5 和 SHA1 散列的连接。对于 DSA 密钥，是摘要的 SHA1 散列。对于匿名密钥交换，不发送此报文。

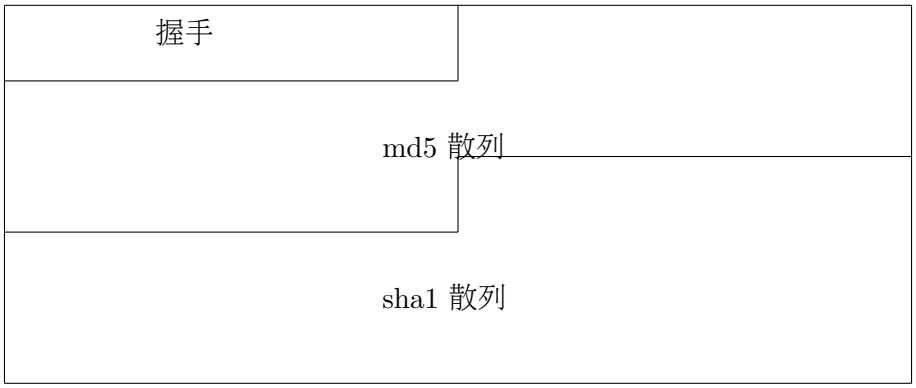
对于 SSLv3，散列计算表述如下：

$$H(master_secret \oplus pad2 \oplus H(digest \oplus master_secret \oplus pad1))$$

其中:
$\oplus$: 是连接
H: 是散列函数，SHA1 或者 MD5
pad1: 48 个字节 0x36(MD5),40 字节 0x36(SHA1)
pad2: 48 字节 0x5C(MD5),40 字节 0x5C(SHA1)

像 TLS 一样，对于 RSA 密钥，MD5 和 SHA1 散列被连接，对于 DSS 密钥，只有 SHA1 散列。匿名密钥交换不发送此报文。

TLSv1 完成报文



4.1.11 完成

在改变密码说明报文之后，服务器和客户端都会发送完成报文。用来确认握手没有被外界干预。一旦接受到了，就在本地重新计算，如果接受到的和本地值不相同，则说明握手有外界干预。第一个报文携带被加密的商定密码族和密钥。

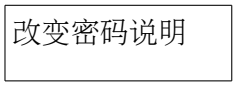
对于 **TLSv1**，确认数据由 **PRF** 产生，有 12 字节。我们将在密钥材料产生部分详细谈到 **PRF**。

PRF 使用主密语作为密语，文本是标签的连接 —对于服务器是 **server**, 对于客户端是 **client**。如果发生了重商谈，这个包含了在之前握手时的每一个握手报文。

对于 **SSLv3**，使用类似 **HMAC** 的嵌套散列。它也使用主密语，握手报文和发送端和客户端标识符。一个独立的类似 HMAC 的散列使用 **MD5** 和 SHA1 生成。然后连接他们作为验证数据。客户端的标识是 0x434C4E54, 服务器是 0x53535652。

SSLv3 和 **TLSv1** 使用标识的原因是使得客户端和服务器的完成保温不相同，防止攻击者发送一个相同的报文到发送端。

4.2 改变密码说明



改变密码说明报文用来通知对端，商定的连接参数在这个报文发送之后立即生效。换句话说，它使得密码族生效。在握手结束后，任何应用程序数据发送之前发送此报文。它仅仅影响连接的方向，因此，只可以是连接的对端发送。

4.3 警告

在连接关闭之前或者有错误发生，警告报文的任一端都可以发送警告报文。警告由级别 — 警告或者严重，和描述构成。这两种值都是 8 比特。描述要么是通知结束或者某些错误状态的描述，比如不是预期的报文，或者记录 mac 损毁。所有的错误藐视在附录 A 中。

4.4 应用程序数据

应用程序数据报文用来在连接的两个端点之间传输数据。它不包含头部，仅仅由数据本身构成，数据封装在记录里。

5 密钥材料生成

密钥材料生成用来产生密钥材料, 作为密钥和输入向量加密验证记录。密钥生成的输入客户端和服务器的随即数，和 pre-master 密语（在客户端和服务器的 hello 报文里）和客户端密钥交换报文。pre-master 密语是发送到服务器的，且是经过自己证书加密的，所以应该只有客户端和服务器知道。

5.1 TLSv1

5.1.1 伪随机函数 (PRF)

在 TLS 中，伪随机函数（**PRF**) 是密钥材料生成的核心。它也用在完成握手报文里。**PRF** 以一个扩展函数开始，扩展函数接受一个密钥和一些文本，产生任意的输出。它定义如下：

$$
\begin{aligned}
P\_H(secret, text) = \quad & HMAC\_H(K, A(1), T) \oplus \\
& HMAC\_H(K, A(2), T) \oplus \\
& HMAC\_H(K, A(3), T) \oplus \cdots
\end{aligned}
$$

其中:

$$
\begin{aligned}
\oplus: \quad & \text{连接} \\
A(): \quad & A(0) = text \\
& A(n) = HMAC\_H(secret, A(n-1)) \\
H: \quad & \text{散列函数,SHA1 或者 MD5} \\
K: \quad & \text{密钥} \\
T: \quad & \text{文本}
\end{aligned}
$$

这个公式要运行多次。要产生 48 字节的输出，对于 **MD5** 要运行 3 次，对于 **SHA1** 要运行 3 次，并且最后 12 个字节要丢弃。

PRF 由运行 **MD5** 的 **PRF** 和运行 **SHA1** 的 **PRF** 异或得到:

$$PRF(secret, text) = P\_MD5(S1, text) \otimes P\_SHA1(S2, text)$$

其中:

$$
\begin{aligned}
\otimes: \quad & \text{异或} \\
S1: \quad & \text{密语的前半部份（按位} \\
S2: \quad & \text{密语的后半部分 (按位)}
\end{aligned}
$$

5.1.2 主密语

客户端和服务器独立的使用 **PRF**, 是为了计算 48 字节的主密语。随后被转换成密钥块（分组）。密钥块尽可能的长，并且被分解为 **MAC** 密语，对成加密密钥和用在对称加密密钥的输入向量。按顺序排序，依次是客户端写 **MAC** 密语，服务器写 **MAC** 密语，客户端写密钥，服务器写密钥，客户端写输入向量，服务器写输入向量。不需要的值被忽略。主密语和密钥块使用 **PRF** 计算如下:

$$
\begin{aligned}
X(secret, label) =& PRF(secret, label \oplus client\_random \oplus sever\_random) \\
master\_secret =& X(pre\_master\_secret, "mastersecret") \\
key\_block =& X(master\_secret, "keyblock")
\end{aligned}
$$

客户端写 MAC 密语	服务器写 密语	客户端写密钥	服务器写密钥	客户端写 IV	服务器写 IV

对于出口密码族，为了服从美国出口限制，输入向量不是继承自主密钥，因为它不是私密的，并且按照出口限制要求，加密密钥从 40 比特扩展。扩展密钥而不是使用本身，是为了让客户端和服务器随机的，防止攻击这从 40 比特密钥空间里查到。

计算方法如下，尽管必须指出美国出口限制并没有放松，因此出口密码族不应该被任何人真正使用。

$$\begin{aligned} final\_client\_write\_key =& X(client\_write\_key, \text{"}clientwritekey\text{"}) \\ final\_server\_write\_key =& X(server\_write\_key, \text{"}serverwritekey\text{"}) \\ iv\_block =& X(0, \text{"}IVblock\text{"}) \end{aligned}$$

输入向量的第一部份由客户端使用，第二部份由服务器端使用。

5.2 SSLv3

SSLv3 主密钥和密钥块计算使用了稍微不同于 **TLS** 的方法。主密钥的长度是 48 字节，密钥块尽可能的长以产生所有的密钥材料。下面的密钥块计算要进可能的多运行，多余的数据丢弃。

$$\begin{aligned} master\_secret =& MD5(pre\_master\_secret \oplus SHA1(\text{"}A\text{"} \oplus \\ & pre\_master\_secret \oplus client\_random \oplus server\_random)) \oplus \\ & MD5(pre\_master\_secret \oplus SHA1(\text{"}BB\text{"} \oplus \\ & pre\_master\_secret \oplus client\_random \oplus server\_random)) \oplus \\ & MD5(pre\_master\_secret \oplus SHA1(\text{"}CCC\text{"} \oplus \\ & pre\_master\_secret \oplus client\_random \oplus server\_random)) \end{aligned}$$

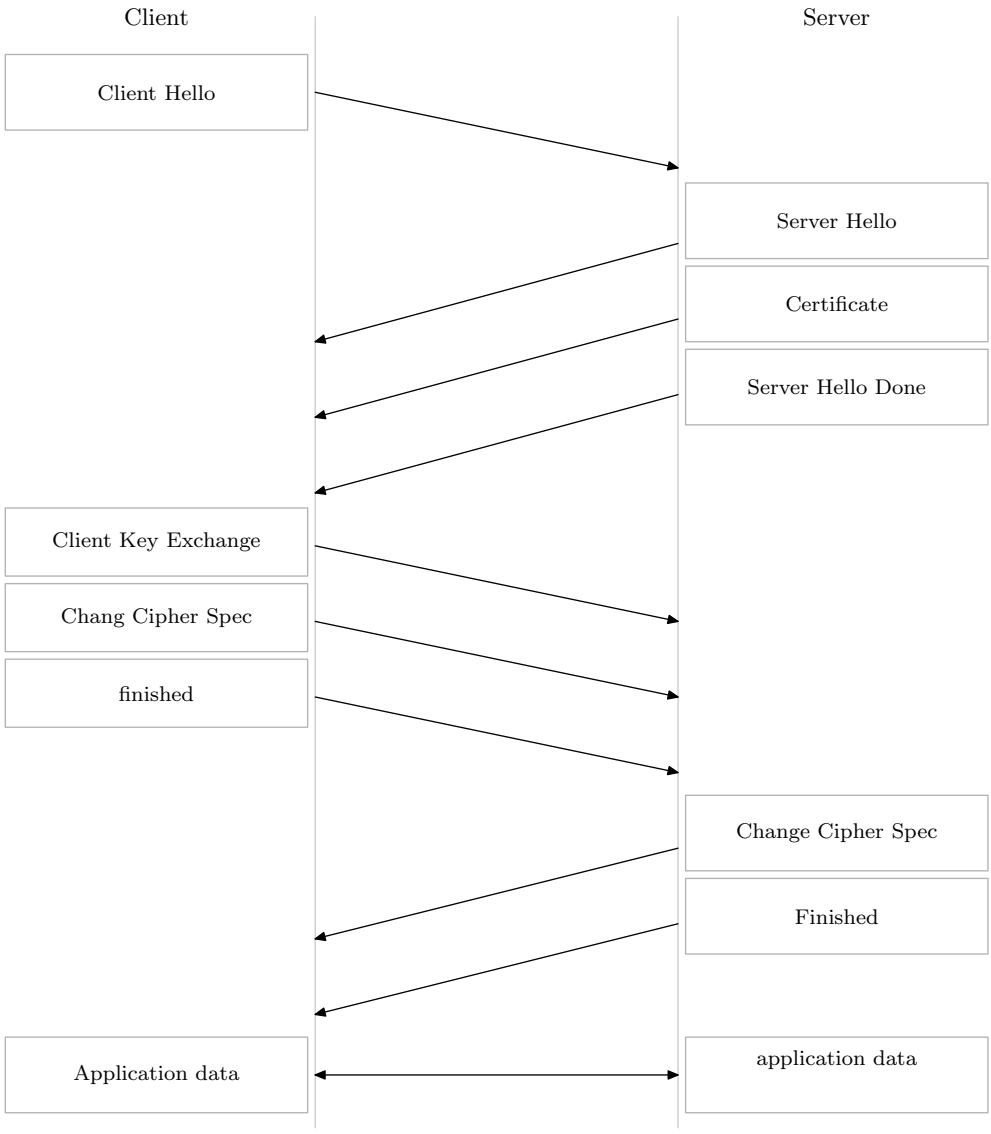
$$\begin{aligned} key_block =& MD5(master\_secret \oplus SHA1(\text{"}A\text{"} \oplus \\ & master\_secret \oplus client\_random \oplus server\_random)) \oplus \\ & MD5(master\_secret \oplus SHA1(\text{"}BB\text{"} \oplus \\ & master\_secret \oplus client\_random \oplus server\_random)) \oplus \\ & MD5(master\_secret \oplus SHA1(\text{"}CCC\text{"} \oplus \\ & master\_secret \oplus client\_random \oplus server\_random)) \end{aligned}$$

正像 **TLS**, 当使用出口密码族时，输入向量必须是非私密的，非对称加密密钥从 40 比特扩展。如下所示：

$$
\begin{aligned}
final_client_write_key &= MD5(client_write_key \oplus client_random \oplus server_random) \\
final_server_key &= MD5(server_write_key \oplus server_random \oplus client_random) \\
client_write_iv &= MD5(client_random \oplus server_random) \\
server_write_iv &= MD5(server_random \oplus client_random)
\end{aligned}
$$

6 握手序列

前面提到的报文协议设计到了许多握手序列。这部份将讲述最常见几个 —**RSA**, 临时 **RSA**,Diffie-Hellman 和会话恢复。其他没有涉及到的序列有客户端认证，定量 Diffie-Hellman 和重商谈。

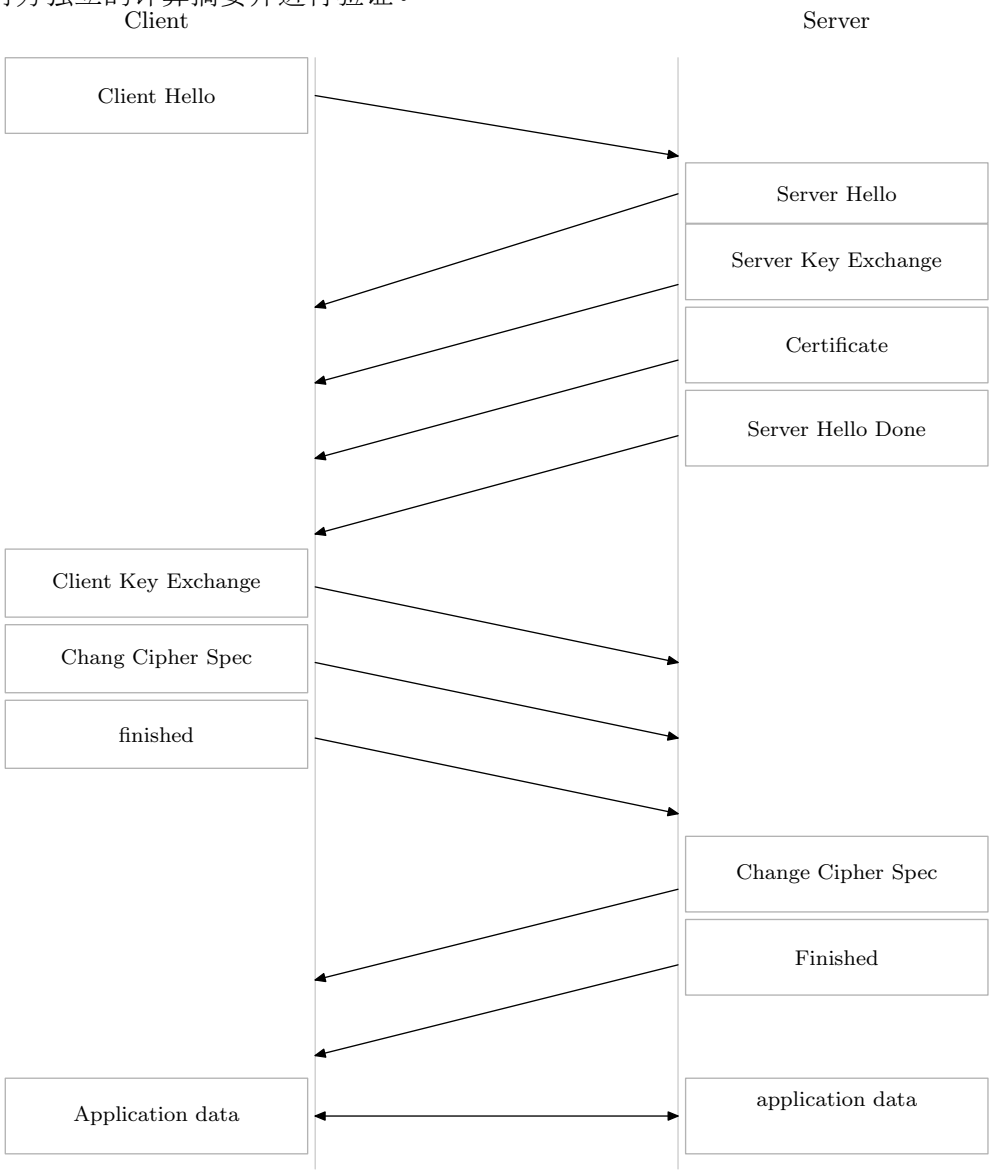


RSA Handshake Sequence

RSA 握手序列用于一个使用非临时 **RSA** 进行密钥交换密码族的回话。在这个握手序列里，两边都计算主密语，使用服务器和客户端随机值，它们分别包含在服务器和客户端 hello 报文里。主密语计算要包含 pre-主密语，它包含在客户端密钥里，用服务器提供的证书报文里的密钥进行加密。

客户端和服务器在发送改变密码说明报文后都立即开始使用上顶的密钥和密码族加密记录。加

密完成的报文包含已经发送和接受的所有握手报文的摘要，这个确认握手没有被第三方破坏。连接的对方独立的计算摘要并进行验证。

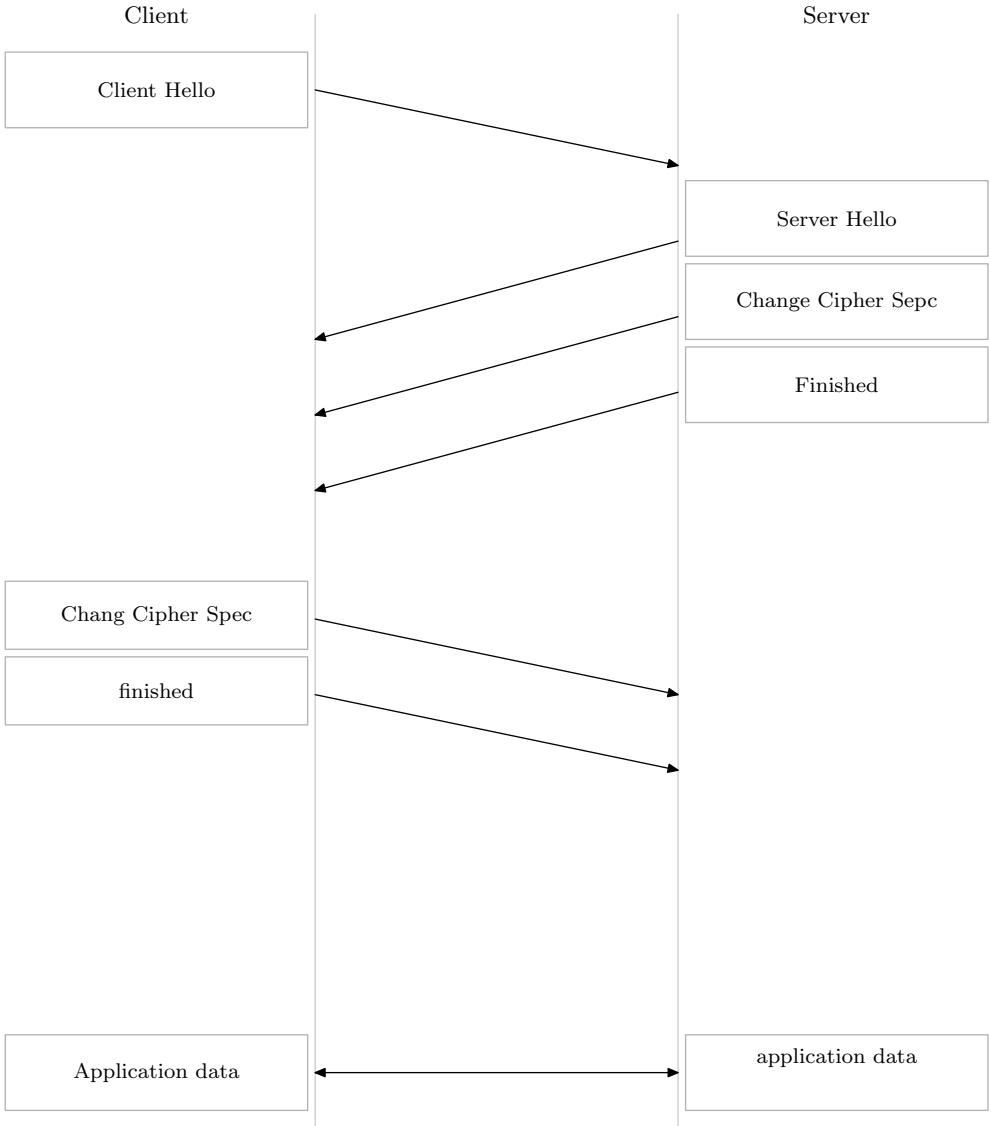


Ephemeral Handshake Sequence

临时握手序列在临时 Diffie-Hellman 和临时 **RSA** 作为密钥交换算法时，使用。它和 **RSA** 的不同点在于，服务器在服务器 hello 报文发送后，立即发送密钥交换报文。这个报文包含了用来加密 pre-master 密语 (在客户端密钥交换报文里）的密钥。

临时 **RSA** 允许用于出口密码的证书长于 512 比特。这是通过只使用证书验证服务器达到的，小于等于 512 比特的临时密钥用来加密客户端密钥交换报文。这是因为出口密码可能不使用长于 512 比特的 **RSA** 密钥加密。但是这个长度的证书很容易受到攻击。

临时 Diffie-Hellman 定义上使用不包含在证书里的临时密钥，并且是在服务器交换报文里发送。



Resumed Session Handshake Sequence

简化握手序列用在会话回复的情况下。这通常发生在如果端点短时间间隔重新连接服务器，比如同一个购物网站访问多个不同的页面。在客户端 hello 报文里，客户端通过之前服务器给定的会话 id 标识自己，服务器在服务器 hello 报文里返回这个会话 id，就隐式表明服务器同意使用简化握手。之前连接的主密语被双方缓存，并在服务器和客户端随机值中使用，在服务器 hello 报文中发送，在客户端 hello 报文中发送。简化握手跳过了服务器解密客户端密钥交换报文获得 pre-master 密语。这个揭秘通常是握手中最慢的部分，除非有硬件加速。也跳过了使用 **PRF** 计算主密语的过程，所以总体上少处理了很多报文，这样握手只需花费 1.5 个往返时延，而不是 2 个。所有这些是的简化握手在大多数情况大速度非常快，对于 **SSL/TLS** 服务器的性能也是非常重要的。

A 警告值

这个附录详细的定义了警告严重性和描述信息，这写都在 **TLSv1** 说明中有定义。

A.1 警告严重性

严重性	数值
警告	1
严重	2

A.2 警告描述

描述	数值
关闭通知	0
非预期报文	10
损坏的记录 mac	20
解密失败	21
记录溢出	22
解压缩失败	30
握手失败	40
证书损坏	42
不支持证书	43
证书重激活	44
证书过期	45
证书不明确	46
非法参数	47
证书不明确	48
禁止访问	49
解码失败	50
解密失败	51
出口限制	60
协议版本	70
不充分安全	71
内部错误	80
用户取消	90
不再重商谈	100