TLS / SSL的工作原理

更新时间：2003年3月28日

适用于：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003 SP1，Windows Server 2003 SP2

TLS / SSL的工作原理

在这个部分

TLS / SSL通过使用基于证书的身份验证和对称加密密钥来验证和保护数据传输。本节讨论如何在Windows Server 2003操作系统中使用RFC标准TLS协议。

本节分为五个小节：

- **Schannel SSP架构**说明了Windows Server 2003中的Microsoft安全支持提供程序接口（SSPI）如何提供可以访问安全通道（Schannel）安全支持提供程序（SSP）的通用例程的机制。
- **TLS / SSL架构**讨论了握手和记录层以及TLS / SSL的相关子协议以及Schannel会话缓存。
- **TLS / SSL协议过程和交互**说明了完整的握手协议，应用程序数据流，恢复安全会话，以及使用TLS消息中包含的内容进行重新协商。某些进程（例如证书映射以及如何处理内部重新协商）是Windows系统特有的，并且在TLS协议的其他实现中可能有或没有不同。
- **TLS / SSL使用的网络端口列表用于TLS / SSL的**网络端口
- **相关信息**列出**相关信息的**链接。

Windows Server 2003操作系统可以使用三种相关的安全协议来通过Internet提供身份验证和安全通信：

- 传输层安全（TLS）1.0
- 安全套接字层3.0
- 安全套接字层（SSL）2.0

所有三个协议通过使用证书和通过各种可能的密码套件的安全通信来提供认证。通用术语*密码套件*是指诸如密钥交换，批量加密和消息完整性的协议的组合。由于身份验证依赖于数字证书，因此Verisign等认证中心（CA）是安全通道（Schannel）的重要组成部分。CA是相互信任的第三方，其确认证书请求者（通常是用户或计算机）的身份，然后向请求者发出证书。证书将请求者的身份绑定到公钥。CA还会根据需要更新和撤销证书。例如，如果向客户端提供服务器的证书，客户端计算机可能会尝试将服务器的CA与客户端的受信任CA列表进行匹配。如果发出的CA是受信任的，客户端将验证证书是可信的并且没有被

篡改。Microsoft Internet Explorer和Internet信息服务（IIS）利用这些协议（最好是TLS）来实现安全超文本传输协议（HTTPS）。

本文档重点介绍TLS，因为TLS正在替换SSL和PCT。TLS在RFC 2246 IETF RFC数据库中标准化。

注意

- 在本文档中，*TLS / SSL*是指传输层安全和安全套接字层的公共协议功能。*TLS*仅指传输层安全性，而*SSL*仅指安全套接字层。

Schannel SSP架构

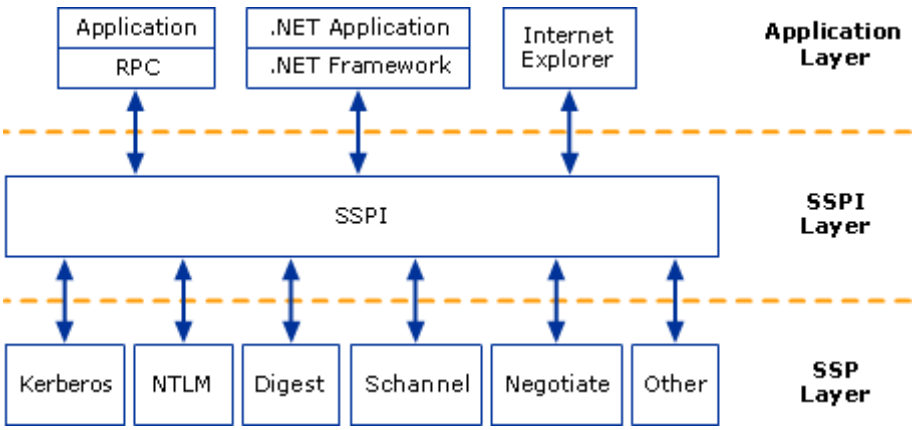
Windows Server 2003操作系统将TLS / SSL协议实现为安全支持提供程序SSP，即操作系统提供的名为Schannel的动态链接库（DLL）。使用哪个SSP取决于连接另一侧的计算机的能力以及正在使用的单个应用程序的配置。

Microsoft安全支持提供程序接口（SSPI）是Windows Server 2003中身份验证的基础。也就是说，需要身份验证的应用程序和基础架构服务使用SSPI提供它。

SSPI是在Windows Server 2003中实现通用安全服务API（GSSAPI）。有关GSSAPI的详细信息，请参阅IETF RFC数据库中的RFC 2743和RFC 2744。

Windows Server 2003 – Kerberos，NTLM，Digest，Schannel和Negotiate身份验证协议中的默认SSP以DLL的形式合并到SSPI中。如果可以与SSPI互操作，则可以并入其他SSP。

SSPI架构



在Windows Server 2003操作系统中，SSPI提供了一种在客户端和服务器之间的现有通信通道上携带身份验证令牌的机制。当两方需要被认证以使得它们可以更安全地通信时，认证请求被路由到SSPI，这完成了认证过程，而不管当前使用的网络协议。SSPI返回透明的二进制大对象，然后在应用程序之间传递这些对象，在这一点上，它们可以传递到该端的SSPI层。因此，SSPI使应用能够使用在计算机或网络上可用的各种安全模型，而不改变到安全系统的接口。

下表描述插入SSPI的SSP组件。在Windows Server 2003中，表中的每个协议以不同的方式使用，以在不安全的网络环境中促进更安全的通信。

SSP层组件

零件	描述
Kerberos V5身份验证	一种行业标准协议，与密码或智能卡一起用于交互式登录。它也是Windows 2000和Windows Server 2003中服务的首选身份验证方法。
NTLM身份验	用于提供与Windows 2000之前的Windows版本的兼容性的质询 – 响应协议。

证	
摘要认证	Windows Server 2003中用于轻量级目录访问协议（LDAP）和Web身份验证的行业标准。摘要作为消息摘要5（MD5）散列或消息摘要在网络上传输凭证。
Schannel	实施SSL和TLS的SSP。Schannel用于跨组织环境中使用的应用程序，例如基于Web的服务器认证，其中用户尝试使用VPN访问安全Web服务器或公司访问。
谈判	可用于协商特定认证协议的SSP。当应用程序调用SSPI以登录到网络时，它可以指定SSP来处理请求。如果应用程序指定协商，协商分析请求，并根据客户配置的安全策略选择最佳SSP来处理请求。

安全通道SSP
您可以使用安全通道（Schannel）SSP访问Web启用的服务，例如在网页上提供的电子邮件或个人信息。

Schannel SSP使用公钥证书来认证各方。它在其套件中包括四个认证协议。当认证方时，它将按照以下优先顺序选择四种协议中的一种：

- TLS版本1.0
- SSL版本3.0
- PCT。PCT在Windows Server 2003中默认关闭。PCT已被安全套接字层3.0和TLS协议取代。Schannel SSP仅支持PCT 1.0的向后兼容性，但此协议在将来的版本中可能不可用。
- SSL版本2.0

然后，Schannel选择双方可以支持的最优选的认证协议。例如，如果服务器支持所有四个Schannel协议，并且客户端仅支持SSL 3.0和PCT，则Schannel使用SSL 3.0进行身份验证。

TLS / SSL架构

Schannel身份验证协议套件基于公钥密码术。该Schannel中套件包括传输层安全（TLS），安全套接字层（SSL）版本3.0，SSL 2.0和专用通信传输（PCT）。所有Schannel协议基于客户端/服务器模型。Schannel客户端向服务器发送消息，服务器响应以验证自身所需的信息。客户端和服务器执行会话密钥的附加交换，并且认证对话结束。当完成认证，安全通信可以使用在认证过程中建立的密钥的服务器和客户端之间开始。

Schannel不需要将服务器密钥存储在域控制器或数据库（如Active Directory）中。但是，客户端必须能够使用可信的授权机构确认凭证的有效性。Schannel使用根CA的证书验证凭据，这些证书在安装Windows Server 2003时加载。因此，用户在验证和创建与服务器的安全连接之前不需要建立帐户。

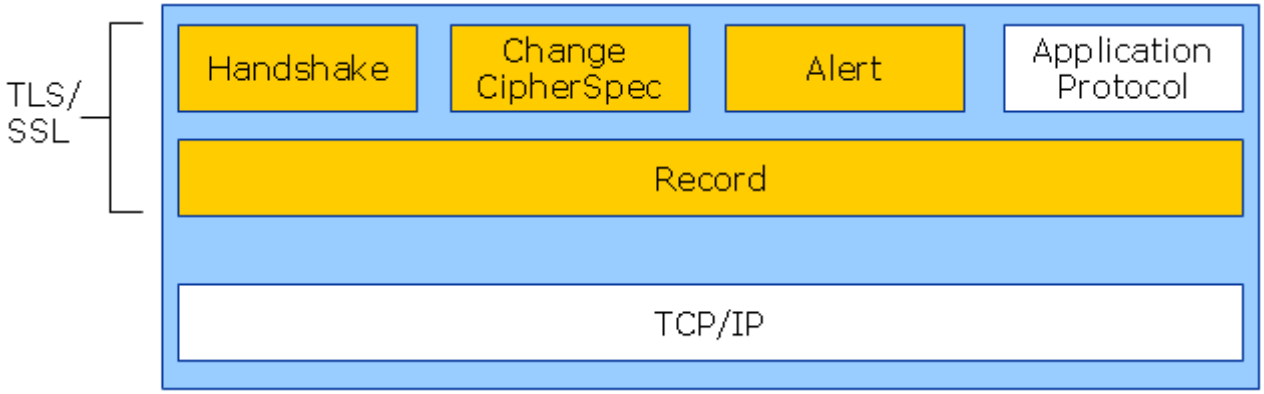
TLS / SSL安全协议在应用协议层和TCP / IP层之间分层，其中它可以保护和发送应用数据到传输层。因为它在应用层和传输层之间工作，TLS / SSL可以支持多个应用层协议。

TLS / SSL假定使用面向连接的传输（通常为TCP）。该协议允许客户端/服务器应用程序检测以下安全风险：

- 消息篡改
- 消息拦截
- 消息伪造

TLS / SSL协议可以分为两层。第一层由应用协议和三个握手子协议组成：握手协议，改变密码规范协议和警报协议。第二层是记录协议。下图说明了各个图层及其组件。

TLS / SSL协议层



握手协议

TLS / SSL协议的握手协议负责建立或恢复安全会话。这一层的主要目标是：

- 协商密码套件和压缩算法。
- 向客户端认证服务器，并且可选地，通过证书和公钥或私钥向服务器认证客户端。
- 交换随机数和预先主秘密。与一些进一步的数据一起，这些值将用于创建记录层将用于散列和加密应用数据的共享秘密密钥。共享密钥称为主密钥。

三个握手子协议是：

- **握手**。该子协议用于在客户端和服务器之间协商会话信息。会话信息包括会话ID，对等体证书，要使用的密码规范，要使用的压缩算法以及用于生成密钥的共享密钥。
- **更改密码规范**。此子协议更改用于在客户端和服务器之间加密的密钥材料。密钥材料是用于创建用于密码使用的密钥的原始数据。Change Cipher Spec子协议由单个消息组成，以通知TLS / SSL会话中的另一方（称为对等方）发送方想要更改为一组新的密钥。该密钥是从握手子协议交换的信息计算的。
- **警报**。该子协议使用消息来向对等体指示状态或错误条件的改变。有各种各样的警报通知对等体正常和错误条件。完整的列表可以在RFC 2246“TLS协议版本1.0”中找到。警告通常在连接关闭，接收到无效消息，消息无法解密或用户取消操作时发送。

握手协议功能

握手协议提供了许多非常重要的安全功能。它执行一组交换，开始认证和协商加密，散列和压缩算法。

验证

证书是通常由认证机构（CA）发布的数字形式的标识，并且包含标识信息，有效期，公钥，序列号和发行者的数字签名。为了认证目的，握手协议使用X.509证书向第二方提供有力证据，帮助证明持有证书的一方的身份和相应的私钥。

CA是相互信任的第三方，其确认证书请求者（通常是用户或计算机）的身份，然后向请求者发出证书。证书将请求者的身份绑定到公钥。CA还会根据需要更新和撤销证书。例如，如果向客户端提供服务器的证书，则客户端计算机可能会尝试将服务器的CA与客户端的受信任CA列表进行匹配。如果发出的CA是受信任的，客户端将验证证书是可信的并且没有被篡改。最后，客户端将接受证书作为服务器身份的证明。

加密

有两种主要的加密类型：*对称密钥*（也称为*共享密钥*）和*不对称密钥*（也称为*公钥*或*公钥 - 私钥*）。TLS / SSL使用对称密钥进行批量加密，使用公钥进行身份验证和密钥交换。

- **对称密钥（也称为私钥）**。在对称密钥加密中，使用相同的密钥来加密和解密消息。如果双方想要安全地交换加密消息，则他们必须都拥有相同对称密钥的副本。对称密钥密码术通常用于加密大量数据，因为其计算速度比非对称密码术快。典型的算法包括数据加密标准（DES），三重DES（3-DES），RC2，RC4和高级加密标准（AES）。

- **不对称密钥，也称为公钥**。非对称密钥加密使用通过复杂的数学过程同时导出的一对密钥。其中一个密钥是公开的，通常要求CA在证书持有者（也称为主体）的证书中发布公钥。私钥由主体保密，从不向任何人透露。密钥一起工作，一个用于执行另一个的逆操作：如果使用公钥加密数据，则只有密钥对中的私钥可以解密; 如果使用私钥进行加密，则必须使用公钥进行解密。这种关系允许公钥加密方案做两个重要的事情。第一，**任何人都可以获得主体的公钥，并使用它来加密只有具有私钥的用户可以解密的数据。第二，如果主体使用其私钥加密数据，任何人都可以通过使用相应的公钥来解密数据。这是数字签名的基础。最常见的算法是Rivest，Shamir和Adleman（RSA）。**

TLS / SSL使用公钥加密来为客户端和客户端（可选）向服务器认证服务器。公钥密码术也用于建立会话密钥。会话密钥用于对称算法中，以更快，更少的处理器密集型对称密钥加密来加密大量数据。

哈希算法

在握手过程中，客户端和服务器同意哈希算法。散列是值到代表值的较小集合的单向映射，使得结果散列的大小小于原始消息，并且散列对于原始数据是唯一的。散列类似于指纹：指纹对个体是唯一的，并且比原始人小得多。散列用于在传输期间建立数据完整性。两种常见的哈希算法是消息摘要5（MD5）和标准哈希算法1（SHA-1）。MD5产生128位散列值，SHA-1产生160位值。

注意

- 可以对数据进行加密和解密，但不能对散列进行逆向工程。散列是一个单向过程。向后运行进程不会重新创建原始数据。这就是为什么计算新的哈希，然后与发送的哈希进行比较。

散列算法包括用于检查所发送的数据的完整性的值。该值通过使用消息认证码（MAC）或散列消息认证码（HMAC）来建立。MAC使用映射函数将消息数据表示为固定长度，优选较小的值，然后对消息进行散列。MAC确保数据在传输期间没有被修改。MAC和数字签名之间的差别在于数字签名也是认证方法。SSL使用MAC。

HMAC类似于MAC，但是使用散列算法与共享秘密密钥的组合。共享密钥附加到要散列的数据。这使得散列更安全，因为双方必须具有相同的共享秘密密钥来证明数据是真实的。TLS使用HMAC。有关HMAC的更多信息，请参阅IETF RFC数据库中的RFC 1024。

更改密码规范协议

更改密码规范协议发信号通知在客户端和服务器之间的连接上使用的密码组的转换。此协议由单个消息组成，该消息使用当前加密套件进行加密和压缩。此消息由具有值1的单个字节组成。此后的消息将使用新的加密套件进行加密和压缩。

警报协议

警报协议包括可从任一方发送的事件驱动警报消息。在警告消息之后，会话结束或者接收者被给予是否结束会话的选择。警报在RFC 2246的TLS规范中定义。下表列出了可能的警报消息及其说明。Schannel SSP将仅根据应用程序的请求生成这些警报消息。

来自警报子协议的事件驱动的警报消息

警报消息	描述
close_notify	通知收件人发件人不会在此连接上再发送任何邮件。
unexpected_message	接收到不适当的消息在正确实现之间的通信中不应该观察到此警报。这个消息总是致命的。
bad_record_mac	接收到具有不正确MAC的记录。这个消息总是致命的。
decrypt_failed	TLSCiphertext记录的解密以无效的方式解密：它不是块长度的偶数倍或其填充值，当检查时，是不正确的。这个消息总是致命的。

record_overflow	接收到长度大于2 ^ 14 + 2048字节的TLSCiphertext记录，或者解密为具有大于2 ^ 14 + 1024字节的TLS压缩记录的记录。这个消息总是致命的。
decompression_failure	从解压缩函数接收不适当的输入，例如将扩展到过长长度的数据。这个消息总是致命的。
handshake_failure	表示发送方在给定可用选项的情况下无法协商可接受的一组安全参数。这是一个致命错误。
bad_certificate	证书有问题，例如，证书已损坏，或者证书包含无法验证的签名。
unsupported_certificate	收到不受支持的证书类型。
certificate_revoked	收到由其签名者撤销的证书。
certificate_expired	收到的证书已过期或当前无效。
certificate_unknown	在处理证书时发生了未指定的问题，使其不可接受。
illegal_parameter	违反的安全参数，例如握手中的字段超出范围或与其他字段不一致。这总是致命的。
unknown_ca	接收到有效的证书链或部分链，但未接受该证书，因为CA证书无法找到或无法与已知的受信任CA匹配。这个消息总是致命的。
拒绝访问	接收到有效的证书，但应用访问控制时，发件人未进行协商。这个消息总是致命的。
decode_error	由于某些字段超出了指定的范围或邮件的长度不正确，因此无法解码邮件。这个消息总是致命的。
decrypt_error	失败的握手密码操作，包括无法正确验证签名，解密密钥交换或验证完成的消息。
export_restriction	检测到不符合出口限制的谈判; 例如，尝试为RSA_EXPORT握手方法传送1024位临时RSA密钥。这个消息总是致命的。
protocol_version	客户端尝试协商的协议版本被识别，但不受支持。例如，出于安全原因可能会避免旧的协议版本。这个消息总是致命的。
insufficient_security	协商失败，特别是因为服务器要求密码比客户端支持的密码更安全。返回而不是handshake_failure。这个消息总是致命的。
内部错误	与对等体无关的内部错误或协议的正确性使其无法继续，例如内存分配失败。该错误与协议无关。这个消息总是致命的。
user_canceled	与协议故障无关的原因取消握手。如果用户在握手完成后取消操作，则更合适的是通过发送close_notify关闭连接。此警报后应紧跟一个close_notify。此消息通常是一个警告。
no_renegotiation	由客户端响应于hello请求发送或由服务器响应于初始握手后的客户端问候发送。这两种情况通常都会导致重新谈判; 当不适当时，收件人应该用此警报响应; 在那时，原始请求者可以决定是否继续进行连接。一种

	适合的情况是服务器已经产生了满足请求的进程; 该进程可能会在启动时接收安全参数（密钥长度，身份验证等），并且在该点之后可能很难传达对这些参数的更改。此消息始终是警告。

记录层

如RFC 2246所指定的，记录层可能有四个功能：

- 它将来自应用程序的数据分割成可管理的块（并重组传入的数据以传递给应用程序）。Schannel SSP不支持在记录层的分段。
- 它压缩数据并解压缩输入数据。Schannel SSP不支持在记录层的压缩。
- 它对数据应用消息认证码（MAC）或散列/摘要，并使用MAC来验证传入数据。
- 它加密散列数据并解密传入数据。

记录协议功能

记录协议从应用层接收和加密数据，并将其传送到传输层。然后，它获取数据，将其分段为适合于密码算法的大小，可选地压缩它（或者，对于接收的数据，解压缩它），应用MAC或HMAC，然后使用在协商期间协商的信息加密（或解密）握手协议。HMAC仅由TLS支持。

然后将所得到的加密数据传递到传输层。

记录层加密状态

记录层在会话建立期间更改加密状态。也就是说，在处理的不同阶段使用不同的加密方法。

- **未使用加密**。在协商加密套件并决定使用哪些安全方法之前，状态为Null。不执行消息认证或加密。
- 使用的公钥**加密**（使用公钥 - 私钥对）。一旦协商加密组并且交换证书，记录层使用适当的MAC（通常是MD5和安全散列算法（SHA-1），如RSA签名所需的那样）对输出数据进行散列，并且使用发送者的私钥对其进行加密。传入数据使用发件人的公钥解密。
- **使用对称密钥加密**（使用共享会话密钥，而不是公共 - 私有密钥对）。在握手序列的结尾附近，握手协议交换允许客户端和服务器计算主秘密的数据，然后导出客户端写MAC密钥，服务器写MAC密钥，客户端写密钥和服务器写密钥它。

  当这些密钥可用时，客户端和服务器向更改密码规范发送消息。此时，客户端和服务器停止使用公共 - 私有密钥对，并开始使用从主密钥导出的共享会话密钥。

TLS / SSL进程和交互

本节提供了TLS / SSL协议的详细说明，特别是握手协议，其关联的消息和警报以及记录协议。

下面列出的前三个操作使用握手协议进行。双方协商他们将使用，验证，交换密钥和交换应用程序数据的加密套件。

握手和密码套件协商

首先，客户端和服务器彼此联系，并选择一个通用的加密套件。套房包括：

- 密钥交换方法，确定如何交换共享主密钥。
- 确定应用程序数据将如何加密的批量加密方法。
- MAC，其确定应用数据将如何被散列和签名以证明完整性。

验证

服务器始终向客户端认证其身份。但是，客户端可能不需要根据应用程序与服务器进行身份验证。认证方法（主要使用哪种数字证书格式）取决于协商的加密套件。

密钥交换

选择加密套件后，客户端和服务器交换用于数据加密的密钥（或创建密钥所需的信息），这又取决于协商的加密套件的要求。密钥交换操作需要以下事项：

- 创建的随机值称为客户端随机和服务器随机。
- 客户端生成一个随机的前主密钥。

客户端必须将前主密钥安全地传输到服务器。

- 使用RSA密钥交换，预主密钥使用服务器的公钥加密。客户端从服务器的证书获取此密钥。
- 使用EDH密钥交换，预主密钥是EDH操作的结果。在这种情况下，临时DH密钥由服务器的私钥签名。

客户端和服务器都使用前主密钥创建共享主密钥。这是通过散列预主秘密以及ClientRandom和ServerRandom值来完成的。主密钥用于创建客户端和服务器共享的四个密钥：

- **客户端写MAC密钥。**此键被添加到客户端消息哈希。客户端使用密钥创建初始散列。服务器使用它来验证客户端消息。
- **服务器写MAC密钥。**此键被添加到服务器消息哈希。服务器使用密钥创建初始散列。客户端使用它来验证服务器消息。
- **客户端写密钥。**客户端使用此密钥对消息进行加密。服务器使用密钥来解密客户端消息。
- **服务器写密钥。**服务器使用此密钥对消息进行加密。客户端使用密钥来解密服务器消息。

最后一个操作在记录层使用记录协议进行。

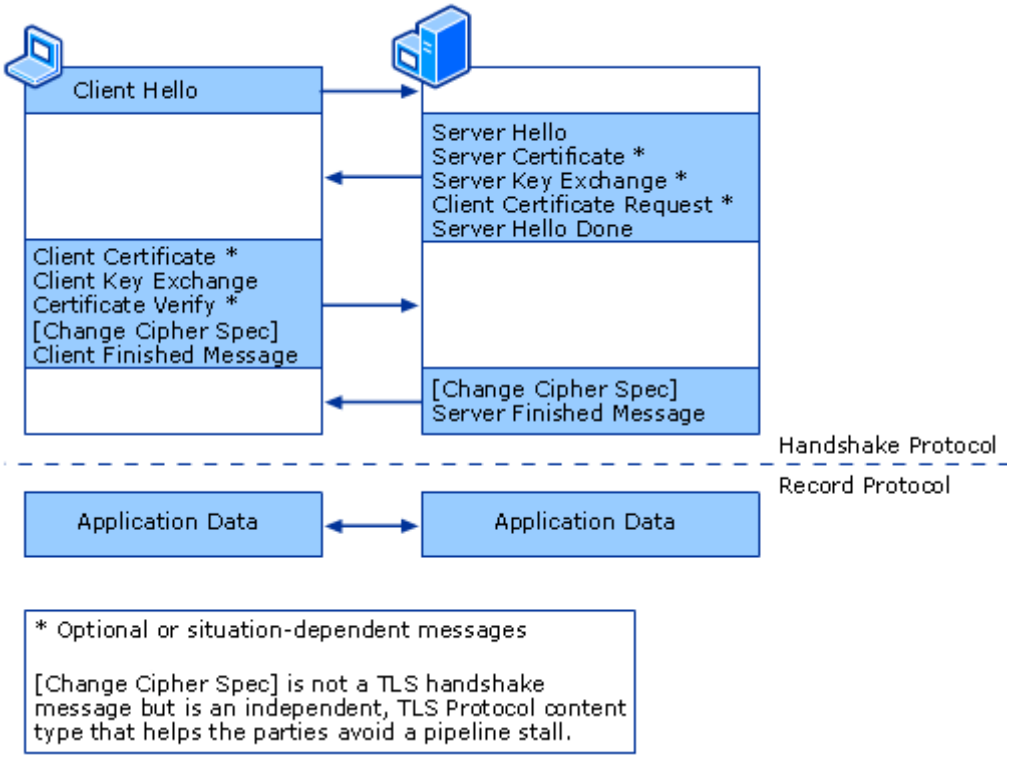
应用程序数据交换

客户端应用程序和服务器应用程序相互通信。所有数据都使用协商的批量加密方法进行加密。

TLS / SSL消息序列概述

使用Schannel认证协议与服务器认证和建立加密信道的过程包括以下步骤：

完全TLS握手



握手协议管理这种交换密钥生成材料的过程，并且它使用握手协议。

一旦哈希和加密密钥准备就绪可以使用，记录层接管。它使用记录协议，使用握手过程中创建的密钥保护应用程序数据。

完全握手协议

握手协议是协商数据传输会话的安全参数的一系列有序消息。

客户端Hello消息

客户端Hello通常是TLS / SSL会话建立序列中的第一个消息。协议允许服务器请求一个hello，但这是应用程序特定的，而不是正常序列的一部分。

客户端Hello消息

客户端Hello消息

客户端通过向服务器发送Client Hello消息来发起会话。客户端Hello消息包含：

- **版本号**。客户端支持的最高版本的版本号。这由客户端发送到服务器。版本2用于SSL 2.0，版本3用于SSL 3.0，版本3.1用于TLS。尽管TLS的IETF RFC是TLS版本1.0，协议在版本字段中使用3.1来表示它是更高版本，具有比SSL 3.0更多的功能。
- **客户端随机**。一个由客户端的日期和时间组成的4字节数字，以及一个28字节的加密生成的伪随机数。这用于计算从其导出加密密钥的主秘密。
- **（可选）会话标识**。用于标识活动或恢复会话状态的字节字符串。使客户机能够恢复先前的会话。恢复先前的会话可能是有用的，因为创建新会话需要处理器密集型公钥操作，这可以通过用已建立的会话密钥恢复现有会话来避免。由SessionID标识的先前会话信息存储在相应的客户端和服务器会话高速缓存中。如果客户端不恢复会话，则此字段为空。
- **密码套件**。客户端上可用的密码套件列表。加密套件的示例是TLS_RSA_WITH_DES_CBC_SHA，其中TLS是协议版本，RSA是将用于密钥交换的算法，DES_CBC是加密算法（在CBC模式中使用56位密钥），并且SHA-1是散列函数。

  注意

- Microsoft应用程序通常指定RSA作为密钥交换算法，RC4作为加密方法，MD5作为消息认证码。

- **压缩算法**。客户端支持的压缩算法。压缩是可选的，并且是在Hello序列完成后记录协议可能对所有数据执行的四个操作之一。

服务器会话缓存允许会话恢复

客户端Hello用于启动新会话或恢复现有会话。如果客户端Hello包括会话ID，则客户端正在尝试恢复会话。

服务器维护会话缓存，以允许最近会话的快速恢复，类似于Kerberos中的票证缓存。恢复会话不需要证书交换，因此它比正常的Hello序列快得多。

应用程序确定客户端是否被允许恢复会话以及空闲会话ID有效的时间。

服务器使用CryptoAPI来管理会话ID和证书缓存。

注意

- 客户端Hello可以在现有会话期间的任何时间发起，并且不限于会话初始化。服务器应用程序可以基于请求的资源来周期性地请求新的Hello以请求客户端认证。客户端应用程序或服务器可能会请求新的握手以刷新加密密钥。

TLS 1.0密码套件

Schannel在TLS 1.0中支持下表中的密码套件。在此表中，使用以下缩写：

CBC =密码块链接

DES =数据加密标准

DHE =短暂的Diffie-Hellman

DSS =数字签名标准

套房按照喜好的顺序列出。

TLS 1.0密码套件

密钥交换	密码	哈希
RSA	RC4 128	MD5
RSA	RC4 128	SHA-1
RSA	3DES EDE CBC	SHA-1
DHE DSS	3DES EDE CBC	SHA-1
RSA	DES CBC	SHA-1
DHE DSS	DES CBC	SHA-1
RSA导出1024	RC4 56	SHA-1
RSA导出1024	DES CBC	SHA-1

DHE DSS导出1024	DES CBC	SHA-1
RSA出口	RC4 40	MD5
RSA出口	RC2 CBC 40	MD5
RSA	没有	MD5
RSA	没有	SHA-1

SSL 3.0密码套件

Schannel支持TLS 1.0密码套件下列出的密码套件。

SSL 2.0密码套件

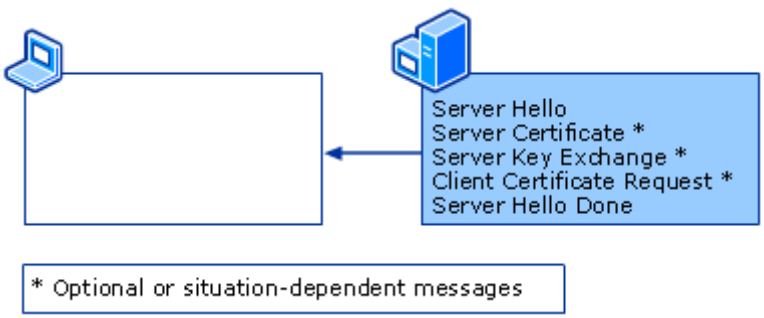
Schannel支持SSL 2.0的以下密码套件。套房按照从最安全到最不安全的顺序列出。

SSL 2.0密码套件

密码	哈希
RC4 128	MD5
DES 192 EDE3 CBC	MD5
RC2 CBC 128 CBC	MD5
DES 64 CBC	MD5
RC4 128 Export 40	MD5

服务器响应客户端Hello

服务器响应客户端Hello



服务器Hello消息

服务器以Server Hello消息响应。服务器Hello消息包括：

- **版本号**。服务器发送双方支持的最高版本号。这是将在连接期间使用的协议版本。
- **服务器随机[32]**。ServerRandom [32]是服务器日期和时间的4字节表示，加上一个28字节的加密生成的伪随机数。客户端和服务器都使用此数字以及客户端随机数来生成将从中导出加密密钥的主密钥。

- **会话标识**（如果有）。这可以是三个选择之一。
    - **新会话ID。**客户端没有指示恢复会话，因此生成了一个新的ID。当客户端指示恢复会话时，还会生成新的会话ID，但服务器无法或不会恢复该会话。
    - **恢复的会话ID**。ID与客户端Hello中指示的相同。客户端指示要恢复的特定会话ID，并且服务器愿意恢复该会话。
    - **Null**。这是一个新的会话，但服务器不愿意在以后恢复它，因此不返回ID。
- **密码套件**。服务器选择客户端和服务器都支持的最强的密码。如果没有双方都支持的密码套件，则会话以*握手失败*警报结束。
- **压缩算法**。如果使用，指定要使用的压缩算法。

服务器证书消息

服务器将其证书发送到客户端。服务器证书包含服务器的公钥。客户端使用此密钥验证服务器并加密Premaster Secret。服务器证书消息包括：

- **服务器的证书列表**。列表中的第一个证书是服务器的X.509v3证书，其中包含服务器的公钥。
- **其他验证证书**。所有其他验证证书，最多但不包括CA的根证书，由CA签名。

客户端还会检查证书中服务器的名称，以验证其是否与客户端用于连接的名称匹配。例如，如果用户在浏览器中键入http://www.contoso.com作为URL，则证书包含主题名称www.contoso.com或* .contoso.com。如果这些名称不匹配，Internet Explorer将警告用户; 其他客户端应用程序通常直接失败连接。

（可选）服务器密钥交换消息

服务器创建临时密钥并将其发送到客户端。客户端可以使用此密钥在过程中稍后对客户端密钥交换消息进行加密。仅当服务器的证书不包含适用于密钥交换的公钥时，或者当密码套件强制使用临时密钥进行密钥交换操作时，才需要执行此步骤。当服务器使用DSS证书时，也会使用此消息。在这种情况下，服务器公钥不适合密钥交换，因此临时DH密钥由服务器创建并在服务器密钥交换消息中发送。

在这些情况下，服务器必须创建和发送使用的临时密钥，而不是服务器的公钥。创建的密钥取决于密码套件。对于不允许大于512位的公钥的RSA的导出版本，临时较短的密钥用不可用的公钥签名以用于真实性。

为什么这很重要？客户端将需要此密钥在客户端密钥交换消息中加密下面讨论的Premaster Secret。

注意

- 此消息不用于Microsoft应用程序的非导出版本，因为非导出RSA证书将始终在其证书中包含服务器的公钥。

（可选）客户端证书请求消息

服务器必须始终向客户端提供其证书，但并不总是要求客户端对其自身进行身份验证。因此，客户端并不总是需要将其证书发送到服务器。如果服务器不需要客户端身份验证，则不会发送此消息。

此步骤可能用于网站，例如银行网站，其中服务器必须在提供敏感信息之前确认客户端的身份。如果应用程序需要相互验证，则服务器发送客户端证书请求。客户端证书请求消息包括：

- 所需证书的类型（通常为RSA或DSS）
- 可接受的CA列表

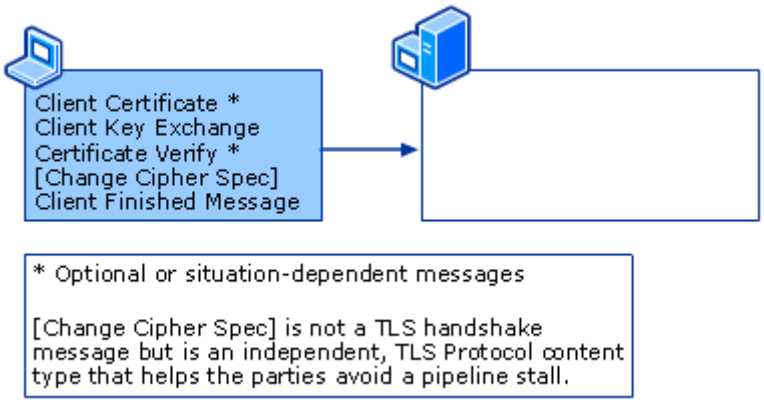
服务器您好完成消息

此消息表示服务器已完成并正在等待客户端的响应。此邮件没有内容。它表示服务器Hello序列完成。

客户端对服务器的响应Hello

客户端接收服务器的消息，并检查服务器证书的认证层次。

客户端对服务器的响应Hello



客户端证书消息（如果请求，则为必需）

如果服务器发送客户端证书请求，客户端将其证书发送到服务器进行客户端认证。客户端的证书包含客户端的公钥。客户端证书消息包括客户端的**证书列表**。列表中的第一个证书是客户端的X.509v3证书，其中包含客户端的公钥。之后是其他验证证书，最多但不包括CA的根证书，由CA签名。

客户端密钥交换消息

客户端在使用在客户端Hello消息和服务器Hello消息期间生成的两个随机值计算预占密钥之后发送客户端密钥交换消息。在将其发送到服务器之前，预授权秘密通过公钥从服务器的证书加密。双方都在本地计算主秘密并从中获得会话密钥。

如果服务器可以解密此数据并完成协议，则客户端确信服务器具有正确的私钥。此步骤对于证明服务器的真实性至关重要。只有具有与证书中的公钥匹配的私钥的服务器才能解密此数据并继续协议协商。

客户端密钥交换消息包括：

- **客户端的协议版本**。服务器将验证它与客户端Hello消息中发送的原始值匹配。此措施防止回滚攻击。回滚攻击通过操作消息来工作，以使服务器和客户端使用较不安全的早期版本的协议。
- **前主秘密**。这是客户端生成的数字（48字节的RSA），用服务器的公钥加密，用于客户端随机和服务器随机创建主密钥。

证书验证消息（如果发送客户端证书，则为必需）

仅当客户端先前发送了客户端证书消息时，才会发送此消息。客户端通过使用其私钥对到目前为止的所有消息的哈希值进行签名来进行身份验证。服务器使用签名者的公钥验证签名，确保签名使用客户端的私钥签名。此消息包含一个长签名，以验证客户端的证书（如果请求并发送）。

对于RSA，签名包括：

- 所有先前握手消息的MD5哈希值。
- 所有先前握手消息的SHA-1散列。
- 两个哈希，它们用客户端的私钥连接和加密。

对于DSS，签名包括：

- 所有先前握手消息的SHA-1散列。

- 使用客户端的私钥加密。

更改密码规范消息

Change Cipher Spec消息通知服务器，包括客户端完成消息的所有未来消息都使用刚刚协商的密钥和算法来加密。在这一点上，客户端和服务器被认证，并且客户端已经发送了预主秘密。客户端和服务器都计算了主密钥。然而，到目前为止，任何加密都使用客户端或服务器的私钥/公钥。Change Cipher Spec消息告诉服务器客户端已准备好使用写密钥进行所有进一步的加密。

注意

- 改变密码规范消息不是握手协议的一部分。TLS RFC为变更密码规范定义了单独的协议。这允许开发人员定义特定的Change Cipher Spec消息。

计算主密钥和后续密钥

握手序列安全地交换用于创建主秘密的数据。然而，这个秘密本身从来不用于加密; 相反，几个密钥是从主密钥导出的。然后，这些密钥用于消息认证和加密。

注意

- 本节和下一节（在记录层中的散列）的细节提供了计算“黑盒子”内部的一瞥，并且意在说明保证通信所必需的复杂性。随着新算法的发展，关键材料交换细节随时间而改变。

计算主秘密

主秘密源自在客户端Hello消息和服务器Hello消息中交换的伪随机素材，该客户端Hello消息和服务器Hello消息先前在客户端和服务器最终消息的部分中讨论。服务器和客户端都创建主密钥。它从不交换。

为了创建主秘密密钥，系统将以下作为变量传递到伪随机函数（PRF）：

- 48字节的前主密钥。
- 文字短语“大师秘密”
- 客户端随机数和服务器随机数的串联。

结果是主秘密。

MAC密钥和写加密密钥

要启用消息加密和散列，需要在服务器和客户端上创建四个密钥：

- 客户端写MAC密钥
- 服务器写MAC密钥
- 客户端写密钥
- 服务器写密钥

这些键从不交换。

为了产生这些密钥，主密钥被传递到PRF，直到产生足够的输出以产生被分为六个部分的值。（最后两个部分仅针对非导出块密码生成。）

客户端写MAC密钥

这个密钥被添加到客户端消息哈希。客户端使用该密钥创建初始散列。服务器使用它来验证客户端消息。

服务器写MAC密钥

此键被添加到服务器消息哈希。服务器使用密钥创建初始散列。客户端使用它来验证服务器消息。

客户端写密钥

客户端使用此密钥对消息进行加密。服务器使用密钥来解密客户端消息。

服务器写密钥

服务器使用此密钥对消息进行加密。客户端使用密钥来解密服务器消息。

完成消息

完成消息是整个会话的散列，其提供对客户端的进一步认证。此消息是一个复杂的验证，发送完成消息的客户端是真正的客户端启动Hello对话。它是记录层使用写入密钥和写入MAC密钥加密和哈希的第一条消息。

在握手协议的消息的构造

为了生成消息内容，系统将以下作为变量传递到伪随机函数（PRF）：

- 主秘密。
- 文字短语“客户完成”。
- 所有先前握手消息的MD5哈希和所有先前握手消息的SHA-1哈希的级联。

握手协议将结果传递到记录层。

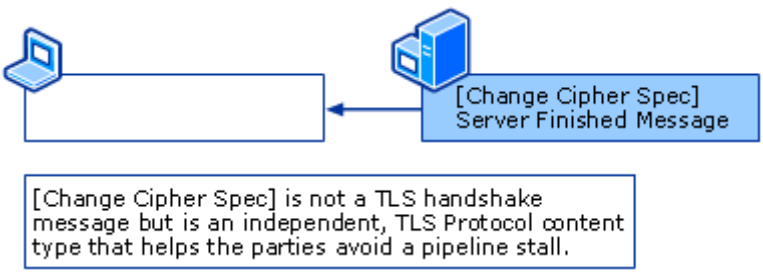
在记录层完成消息

记录层使用HMAC与从主密钥导出的客户端写MAC密钥来哈希数据。然后，记录层使用客户端写密钥加密数据，客户端写密钥也是从主密钥导出的。

服务器完成的消息

如果服务器具有其公开密钥对的私有半边的副本，则它可以解密ClientKeyExchange消息并生成匹配的加密和MAC密钥集。服务器用与秘密密钥密码术通信的请求来响应客户端，使得双方已经确认他们将使用他们已经同意的密码和密钥。

服务器完成的消息



更改密码规范消息

此消息通知客户端服务器将开始使用刚刚协商的密钥加密消息。服务器使用会话密钥将其记录层安全状态切换为对称加密。

完成消息

此消息是使用会话密钥和MAC密钥的此点的整个交换的散列。如果客户端可以成功解密此消息并验证包含的散列，客户端确保TLS / SSL握手成功，并且在客户端计算的密钥与在服务器上计算的密钥匹配。

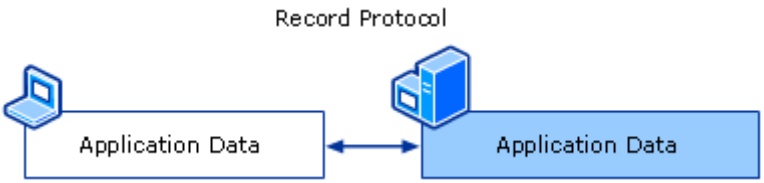
此消息与客户端完成消息相同，但有以下区别：

- 使用文字短语“服务器完成”而不是“客户端完成”。
- 散列握手消息包括客户端的客户端完成消息。
- 服务器使用服务器写MAC密钥和服务器写密钥在记录层进行散列和加密。

应用程序数据流

到目前为止，在这个过程中，客户端和服务器应用程序已通过使用证书和公钥/私钥进行身份验证，Premaster Secret和其他数据已交换以创建主密钥，并且服务器和客户端都已成功证明它们在整个Hello序列中没有改变标识。现在应用程序可以使用已建立的键和参数开始通信。记录层分段，压缩，散列和加密所有进一步的通信。在必要时，它解密，验证，解压缩和重新组装通信

记录协议



记录协议

当记录协议从应用层接收数据时，它可以执行以下任务：

- 将数据分段为块或将分段的数据重组为其原始结构。Schannel不支持记录层的碎片。
- 对消息中的数据块序列进行编号，以防止尝试对数据重新排序的攻击。
- 使用握手协议中协商的压缩算法压缩或解压缩数据。Schannel不支持在记录层的压缩。
- 使用在握手协议期间协商的加密密钥和加密算法来加密或解密数据。
- 将HMAC（或对于SSL 3.0，MAC）应用于传出数据。然后它计算HMAC并验证它与发送的值相同，以便在接收到消息时检查数据完整性。

一旦记录协议完成了对数据的操作，它就将数据发送到应用程序进行传输。如果数据传入，则将其发送到适当的进程进行接收。

HMAC哈希在记录层

分段的压缩应用程序数据在被加密之前经过记录层时被散列。

注意

- 本节中的细节提供了计算“黑盒子”内部的一瞥，并且意在说明保证通信所必需的复杂性。随着新算法的发展，关键材料交换细节随时间而改变。

MAC由散列的两个值组成：

- 密钥值：客户端或服务器写入MAC密钥（从主密钥派生）。
- 连接：
  - 记录的序列号。
  - 包类型。

- 协议版本。
- 包长度。
- 未加密的应用程序数据。

然后将该散列的结果和原始压缩数据片段加密并传递给TCP。

当消息在其目的地解密时，基于压缩片段和MAC密钥计算新的散列。将新的哈希与在消息中发送的哈希进行比较。如果它们匹配，则验证消息的数据完整性。

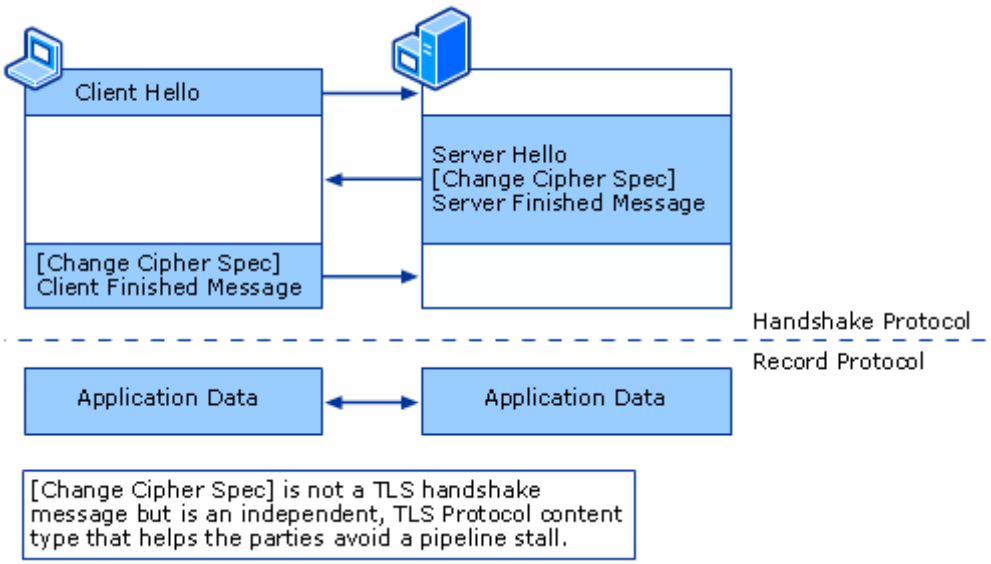
注意

- 可以对数据进行加密和解密，但不能对散列进行逆向工程。散列是一个单向过程。向后运行进程不会重新创建原始数据。这就是为什么计算新的哈希，然后与发送的哈希进行比较。

恢复安全会话

一旦在客户端和服务器之间建立了安全会话，客户端可以使用在先前会话中使用的相同密钥来尝试在将来恢复会话。如果服务器能够恢复会话，则将发生下面的握手协议的简化版本。如果不是，则将发生完全握手。

恢复会话消息



- 客户端使用要恢复的会话的会话ID发送客户端问候消息。
- 服务器检查其会话缓存中是否有匹配的会话ID。如果找到匹配并且服务器能够恢复会话，则它发送具有会话ID的服务器Hello消息。

  注意

  - 如果未找到会话ID匹配，则服务器生成新的会话ID，并且TLS客户端和服务器执行完全握手。
- 服务器必须发送更改密码规范消息以通知客户端服务器将开始使用一组新的加密和MAC密钥来加密消息，这基于来自当前连接的缓存的主秘密和客户端随机服务器随机值。这确保每个连接都有自己的一组加密密钥。服务器使用会话密钥将其记录层安全状态切换为对称加密。
- 服务器发送服务器完成消息。
- 客户端必须发送Change Cipher Spec消息以通知服务器客户端将开始使用会话密钥进行散列和加密消息。
- 客户端发送客户端完成消息。

- 客户端和服务器现在可以通过安全通道恢复应用程序数据交换。

重新谈判

重新谈判方法

重新协商可以源自客户端或服务器，但以下情况除外：

- 在Windows 2000和Windows XP中无客户端重新协商。
- SSL 2.0中无重新协商。

服务器重新协商场景

服务器可以在两种情况下请求重新协商：

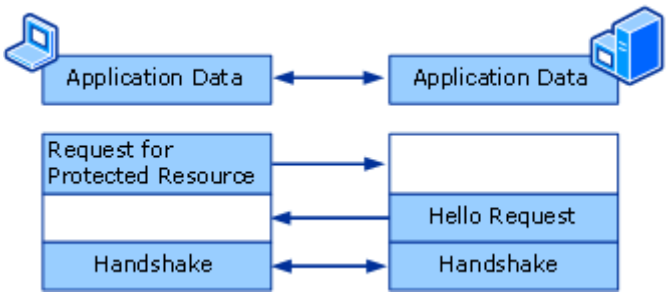
- 在初始握手之后，客户端请求访问受保护的资源。服务器可以发起重新协商以请求客户端认证。这是服务器发送Hello请求的情况。当客户端接收到Hello请求时，它通过发送客户端Hello消息开始新的握手。
- 刷新键。这以与上述第二种情况相同的方式执行。

注意

- Schannel不允许客户端忽略Hello请求。客户端必须通过发送客户端Hello消息来启动新的握手，否则服务器将关闭具有致命错误的连接。

典型的服务器重协商

服务器启动重新协商



服务器启动的重新协商使用以下过程：

- 客户端和服务器成功完成完整的TLS / SSL握手。
- 客户端请求访问受保护的资源。
- 服务器向客户端发送Hello Request消息。
- 客户端接收到Hello Request消息。

  注意

  - Schannel不允许客户端忽略Hello请求，但是其他客户端可能。客户端必须通过发送客户端Hello消息来启动新的握手，否则Windows服务器会关闭连接。

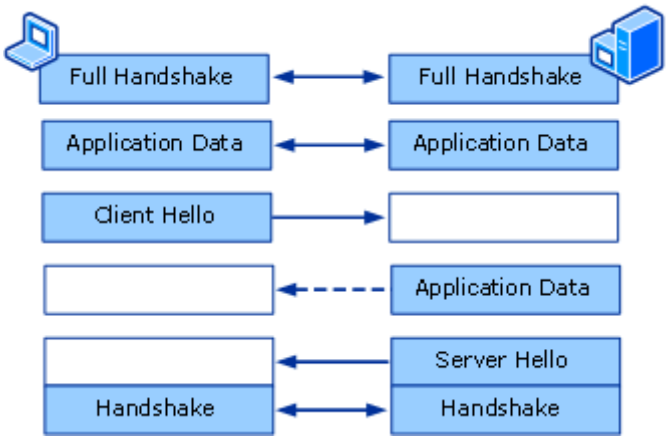
- 客户端向服务器发送客户端问候消息以发起新的完全握手。

客户端重新协商场景

客户端还可以通过发送客户端问候消息随时发起重新协商。通常执行此操作以刷新加密密钥。

典型客户端重新协商

客户端启动重新协商



客户端启动的重新协商使用以下过程：

- 客户端和服务器成功完成完整的SSL握手。
- 客户端需要刷新加密密钥。客户端向服务器发送Client Hello消息。
- 服务器接收客户端问候消息。服务器无法知道此消息不是应用程序数据，因此它传递消息。它作为包含未处理的客户端问候消息的“额外”缓冲区返回，其具有表示它是重新协商的标志。

  注意

  - 服务器可能不会立即响应重新协商请求。在最一般的情况下，服务器可能已经在客户端请求重新协商的同时发送了一些应用数据，或者它可能在响应重新协商请求之前决定发送一些数据。但是，客户端在发起重新协商后将不会再发送更多应用程序数据。
- 服务器向客户端发送服务器Hello消息组。
- 客户端接收服务器Hello消息。客户端无法知道此消息不是应用程序数据，因此它传递消息。它作为包含未处理的服务器问候消息的“额外”缓冲区返回，其具有表示它是重新协商的标志
- 客户端发送下一组握手消息，它们被发送到服务器，握手如常进行。

Schannel会话缓存

客户端第一次连接到服务器时，将执行完全握手。一旦完成，主秘密，密码套件和证书存储在相应的客户机和服务器机器上的会话高速缓存中。TLS / SSL支持重新连接操作，可用于使客户端和服务器恢复先前协商的会话。这允许同一客户端和服务器之间的后续连接使用重新连接握手，而不是完全握手。这在许多情况下是期望的，因为TLS / SSL重新连接需要少得多的CPU和网络带宽，因为不需要密钥交换或证书交换。但有些情况下重新连接可能很麻烦; 例如在性能测试期间或客户端从不重新连接到同一Web服务器时。

Schannel通过使用Schannel会话缓存支持重新连接，因为它保留了使用安全主体用来建立其身份的当前已验证登录数据建立的先前TLS / SSL会话的列表。这些会话由TLS / SSL会话ID引用。Schannel当前最多可维护一万个缓存会话，最多可维持十个小时。

Schannel如何使用证书映射

当服务器应用程序需要客户端认证时，Schannel会自动尝试将客户端提供的证书映射到用户帐户。您可以通过创建将证书信息与Windows用户帐户相关联的映射来验证使用客户端证书登录的用户。创建并启用证书映射后，每次客户端提交客户端证书; 您的服务器应用程序会自动将该用户与相应的Windows用户帐户关联。

映射类型

在大多数情况下，证书以两种方式之一映射到用户帐户：单个证书映射到单个用户帐户（一对一映射），或多个证书映射到一个用户帐户（多到 - 一个映射）。默认情况下，schannel将按照首选顺序使用以下四种证书映射方法。

S4U证书映射

这是Windows Server 2003的新增功能。Schannel使用来自客户端证书的UPN，并尝试使用Kerberos安全软件包进行S4U登录。颁发客户端证书的CA必须具有NTAuth信任。服务器的域和用户必须在Windows Server 2003功能级别上运行。这是唯一的映射方法，即使服务器位于与客户端用户不同的林中也是如此。

用户主体名称映射

企业CA在每个证书中放置一个称为用户主体名称（UPN）的条目。UPN看起来非常像一个电子邮件名称。UPN在Active Directory林中是唯一的。UPN在Active Directory中查找用户的帐户，并且该帐户已登录。UPN映射是隐式的，因此不需要管理员创建和维护。

Schannel使用来自客户端证书的UPN，并查询Active Directory以查找匹配的用户帐户。颁发客户端证书的CA必须具有NTAuth信任。

一对一映射也称为主题/发行者映射

将单个用户证书映射到单个Active Directory用户帐户的方法。与UPN映射不同，管理员必须将证书显式映射到用户的帐户。此证书可以来自任何来源，只要该证书的根CA是受信任的客户端身份验证。客户端证书可以更新等，而不影响此过程，只要主题和颁发者名称字段不更改。

例如，假设用户从您公司批准的CA获取证书。然后获取这些用户证书并将其映射到Active Directory用户帐户。这允许用户通过提供其客户端证书从任何地方使用TLS连接到服务器。

Schannel builds an “altSecurityId” string from the client’s certificate’s subject and issuer name fields, and queries Active Directory for a user account with a matching **altSecurityId** property.

Many-to-One Mapping

Many-to-one mapping involves mapping many certificates to a single user account. Many-to-one mapping works the same way as one-to-one mapping, but it maps all client certificates issued by a given CA to the same user account. This certificate can come from any source, as long as the root CA for that certificate is trusted for client authentication.

Schannel builds an “altSecurityId” string from the client’s certificate’s issuer name field, and queries Active Directory for a user account with a matching **altSecurityId** property.

Network Ports Used by TLS/SSL

通过TLS / SSL的通用应用程序的端口分配

服务名称	TCP
smtp	25
https	443
nntps	563
ldaps	636
ftps数据	989

ftps	990
telnets	992
imaps	993
pop3s	995
ms-sql-s	1433
mfst-gc-ssl	3269
tftps	3713

相关信息

- RFC 2246在IETF RFC数据库
- IETF RFC数据库中的RFC 3546
- IETF RFC数据库中的RFC 2817

社区添加

HMAC引用是错误的

HMAC是RFC 2104而不是RFC 1024

吉尔维德

2/24/2016