Windows Server 2003

SSL/TLS in Windows Server 2003

Chris Crall, Mike Danseglio, and David Mowers

Microsoft Corporation

Published: July 2003

Abstract

The Microsoft® Windows Server™ 2003 operating system supports Schannel, an implementation of three protocols (TLS 1.0, SSL 3.0 & SSL 2.0) that provide network security for applications and services. This paper focuses on the Secure Sockets Layer (SSL 3.0) protocol and Transport Layer Security (TLS 1.0) protocol. These protocols offer certificate-based authentication and secure data transfers using symmetric encryption keys.

Introduction

How can you secure data being sent between applications across an untrusted network? The Security Support Provider Interface (SSPI) in the Microsoft® Windows Server™ 2003 operating system provides one answer to this very common security question. SSPI is an application interface that provides the security services for Windows Server 2003. SSPI supports Schannel, an interface that implements three industry standard protocols:

- Transport Layer Security (TLS) version 1.0
- Secure Sockets Layer (SSL) version 3.0
- Secure Sockets Layer (SSL) version 2.0

SSL/TLS is most widely recognized as the protocol that provides secure HTTP (HTTPS) for internet transactions between Web browsers and Web servers. It can also be used for other application level protocols such as FTP, LDAP, and SMTP. The SSL/TLS protocol enables server authentication, client authentication, data encryption and data integrity over networks such as the World Wide Web.

SSPI is documented in the Microsoft Platform Software Development Kit (SDK), which also includes sample applications[1].

History and Standards for SSL and TLS

SSL was developed by Netscape Communications Corporation in 1994 to secure transactions over the World Wide Web. Soon after, the Internet Engineering Task Force (IETF) began work to develop a standard protocol to provide the same functionality. SSL 3.0 was used as the basis for that work, which is known as the Transport Layer Security protocol (TLS). The implementation of the SSL/TLS protocol in Windows Server 2003 closely follows the specification defined in RFC 2246, "The TLS Protocol Version 1.0."

Differences Between SSL and TLS

Although there are some slight differences between SSL 3.0 and TLS 1.0, this paper will refer to the protocol as SSL/TLS. One important difference is that TLS 1.0 applies a Keyed-Hashing for Message Authentication Code (HMAC) algorithm, whereas SSL 3.0 applies the Message Authentication Code (MAC) algorithm. The HMAC produces an integrity check value as the MAC does, but with a hash function construction that makes the hash much harder to break. For more information about the HMAC, see "The Handshake Protocol and Public Key Encryption" later in this paper.

> **Note** Although their differences are minor, TLS 1.0 and SSL 3.0 do not interoperate. If the same protocol is not supported by both parties, the parties must negotiate a common protocol to communicate successfully.

Benefits of SSL/TLS

SSL/TLS provides numerous benefits to clients and servers, including:

Strong authentication, message privacy, and integrity. The primary feature of SSL/TLS is the ability to secure transmitted data using encryption. SSL/TLS also offers server authentication and, optionally, client authentication to prove the identities of parties engaged in secure communication. It also provides data integrity through an integrity check value. In addition to protecting against data disclosure through encryption, the SSL/TLS security protocol can be used to protect against masquerade attacks, man-in-the-middle or bucket brigade attacks, rollback attacks, and replay attacks.

Interoperability. SSL/TLS works with most Web browsers, including Microsoft Internet Explorer and Netscape Navigator, and on most operating systems and Web servers including the Microsoft® Windows operating system, UNIX, Novell, Apache (version 1.3 and later), Netscape Enterprise Server, and Sun Solaris. In addition, it is often integrated in news readers, LDAP servers, and a variety of other applications.

Algorithm flexibility. SSL/TLS provides options for the authentication mechanisms, encryption algorithms, and hashing algorithms that will be used during the secure session.

Ease of deployment. SSL/TLS is used transparently by many applications on Windows Server 2003. Using SSL for secure browsing when using Internet Explorer and Internet Information Services (IIS) is as easy as selecting a check box.

Ease of use. Because SSL/TLS is implemented beneath the application layer, most of its operations are completely invisible to the client. This allows the client to have little or no knowledge of secure communications and still be protected from attackers.

Drawbacks of SSL/TLS

There are a few drawbacks to using SSL/TLS, including:

Increased processor load. This is the most significant drawback to implementing SSL/TLS. Cryptography, specifically public key operations, are CPU intensive. As a result, there is a performance penalty when using SSL. Unfortunately, there is no single answer to the frequently asked question: how much of performance penalty? The penalty varies widely depending on how often connections are established and how long they last. The greatest overhead occurs while connections are being set up.

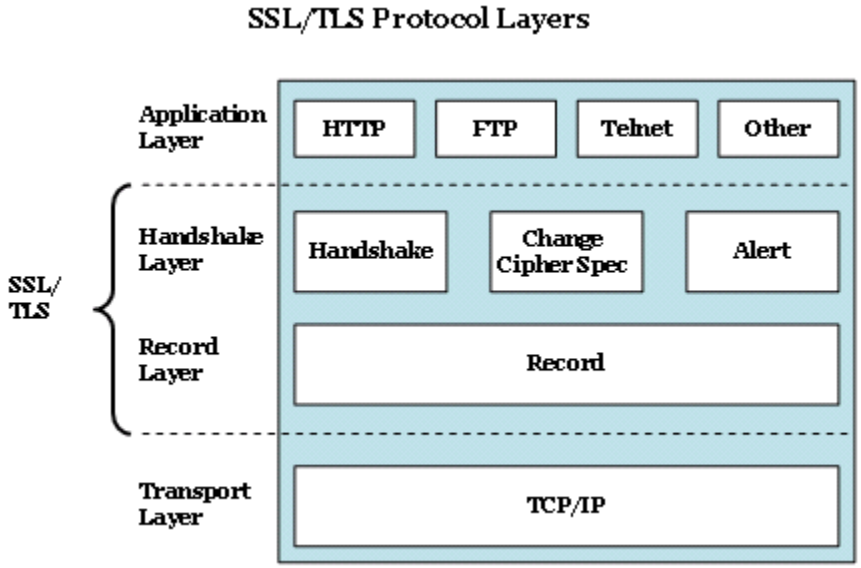
Administrative overhead. An SSL/TLS environment is complex and requires maintenance; the system administrator needs to configure the system and manage certificates.

Overview of SLS/TLS Encryption

SSL/TLS is primarily used to encrypt confidential data sent over an insecure network such as the Internet. In the HTTPS protocol, the types of data encrypted include the URL, the HTTP header, cookies, and data submitted through forms. A Web page secured with SSL/TLS has a URL that begins with "https://".

The SSL/TLS security protocol is layered between the application protocol layer and TCP/IP layer, where it can secure and then send application data to the transport layer. Because it works between the application layer and the TCP/IP layer, SSL/TLS can support multiple application layer protocols.

The SSL/TLS protocol can be divided into two layers. The first layer is the Handshake Protocol Layer, which consists of three sub-protocols: the Handshake Protocol, the Change Cipher Spec Protocol, and the Alert protocol. The second layer is the Record Protocol Layer. Figure 1 illustrates the various layers and their components.



If your browser does not support inline frames, click here to view on a separate page.

Figure 1 SSL/TLS Protocol Layers

The Handshake Layer

The Handshake Layer consists of three sub-protocols:

- **Handshake.** This sub-protocol is used to negotiate session information between the client and the server. The session information consists of a session ID, peer certificates, the cipher spec to be used, the compression algorithm to be used[2], and a shared secret that is used to generate keys.

- **Change Cipher Spec.** The Change Cipher Spec sub-protocol is used to change the *keying material* used for encryption between the client and server. Keying material is raw data that is used to create keys for cryptographic use. The Change Cipher Spec sub-protocol consists of a single message to tell other party in the SSL/TLS session, who is also known is the peer, that the sender wants to change to a new set of keys. The key is computed from the information exchanged by the Handshake sub-protocol.
- **Alert.** Alert messages are used to indicate a change in status or an error condition to the peer. There are a wide variety of alerts to notify the peer of both normal and error conditions. A full list can be found in RFC 2246, "The TLS Protocol Version 1.0.". Alerts are commonly sent when the connection is closed, an invalid message is received, a message cannot be decrypted, or the user cancels the operation.

Handshake Sub-Protocol Functions

The Handshake sub-protocol provides a number of very important security functions. It performs a set of exchanges that starts authentication and negotiates the encryption, hash, and compression algorithms.

Authentication

For authentication purposes, the Handshake Protocol uses an X.509 certificate[3] to provide strong evidence to a second party that helps prove the identity of the party that holds the certificate and the corresponding private key. A certificate is a digital form of identification that is usually issued by a certification authority (CA) and contains identification information, a validity period, a public key, a serial number, and the digital signature of the issuer.

A CA is a mutually trusted third party that confirms the identity of a certificate requestor (usually a user or computer), and then issues the requestor a certificate. The certificate binds the requestor's identity to a public key. CAs also renew and revoke certificates as necessary. For example, if a client is presented with a server's certificate, the client computer might try to match the server's CA against the client's list of trusted CAs. If the issuing CA is trusted, the client will verify that the certificate is authentic and has not been tampered with. Finally, the client will accept the certificate as proof of identity of the server.

For more information about certificates and how they are used, see the Certificates documentation.

Note Windows Server 2003 includes software that allows it to function as a certification authority. For more information, see Help and Support Center in Windows Server 2003.

Encryption

There are two main types of encryption: symmetric key (also known as shared secret key) and asymmetric key (also known as public key or public-private key). SSL/TLS uses both symmetric key and asymmetric key encryption.

- **Symmetric Key.** In symmetric key encryption, the same key is used to encrypt and decrypt the message. If two parties want to exchange encrypted messages securely, they must both possess a copy of the same symmetric key. Symmetric key cryptography is often used for encrypting large amounts of data because it is computationally faster than asymmetric cryptography. Typical algorithms include DES (Data Encryption Standard), 3-DES (Triple DES), RC2, RC4, and AES (Advanced Encryption Standard).
- **Asymmetric Key.** Asymmetric or public key encryption uses a pair of keys that have been derived together through a complex mathematical process. One of the keys is made public, typically by asking a CA to publish the public key in a certificate for the certificate-holder (also called the *subject*). The private key is kept secret by the subject and never revealed to anyone. The keys work together, with one being used to perform the inverse operation of the other: If the public key is used to encrypt data, only the private key of the pair can decrypt it; if the private key is used to encrypt, the public key must be used to decrypt. This relationship allows a public key encryption scheme to do two important things. First, anyone can obtain the public key for a subject and use it to encrypt data that only the user with the private key can decrypt. Second, if a subject encrypts data using its private key, anyone can decrypt the data by using the corresponding public key. This is the foundation for digital signatures. The most common algorithm is RSA (Rivest, Shamir & Adleman).

SSL/TLS uses public key encryption to authenticate the server to the client, and optionally the client to the server. Public key cryptography is also used to establish a *session key*. The session key is used in symmetric algorithms to encrypt the bulk of the data. This combines the benefit of asymmetric encryption for authentication with the faster, less processor-intensive symmetric key encryption for the bulk data.

Hash Algorithms

During the Handshake process the hash algorithm is also agreed upon. A hash is a one-way mapping of values to a smaller set of representative values, so that the size of the resulting hash is smaller than the original message and the hash is unique[4] to the original data. A hash is similar to a fingerprint: a fingerprint is unique to the individual and is much smaller than the original person. Hashing is used to establish data integrity during transport. Two common hash algorithms are Message Digest 5 (MD5) and Standard Hash Algorithm 1 (SHA-1). MD5 produces a 128 bit hash value and SHA-1 produces a 160 bit value.

The hash algorithm includes a value used to check the integrity of the transmitted data. This value is established using either a MAC or an HMAC. The MAC uses a mapping function to represent the message data as a fixed-length, preferably smaller, value and then hashes the message. The MAC ensures that the data has not been modified during transmission. The difference between a MAC and a digital signature is that a digital signature is also an authentication method. SSL uses a MAC.

The HMAC is similar to the MAC but uses a hash algorithm in combination with a shared secret key. The shared secret key is appended to the data to be hashed. This makes the hash more secure because both parties must have the same shared secret key to prove the data is authentic. TLS uses an HMAC. For more information about HMAC see RFC 1024, "Keyed-Hashing for Message Authentication."

For more information about public key cryptography, see "Designing a Public Key Infrastructure" in *Designing and Deploying Directory and Security Services* of the *Microsoft® Windows® Server 2003 Deployment Kit*, (or see "Designing a Public Key Infrastructure").

The Record Layer

The protocol at the record layer receives and encrypts data from the application layer and delivers it to the Transport Layer. The Record Protocol takes the data, fragments it to a size appropriate to the cryptographic algorithm, optionally compresses it[5] (or, for data received, decompresses it), applies a MAC or HMAC (HMAC is supported only by TLS) and then encrypts (or decrypts) the data using the information negotiated during the Handshake Protocol.

SSL/TLS in Detail

This section provides a detailed explanation of the SSL/TLS protocol, specifically the handshake protocol, its associated messages and alerts, and the record protocol.

The Handshake Protocol

The handshake protocol is a series of sequenced messages that negotiate the security parameters of a data transfer session. Figure 2 illustrates the message sequence in the handshake protocol.

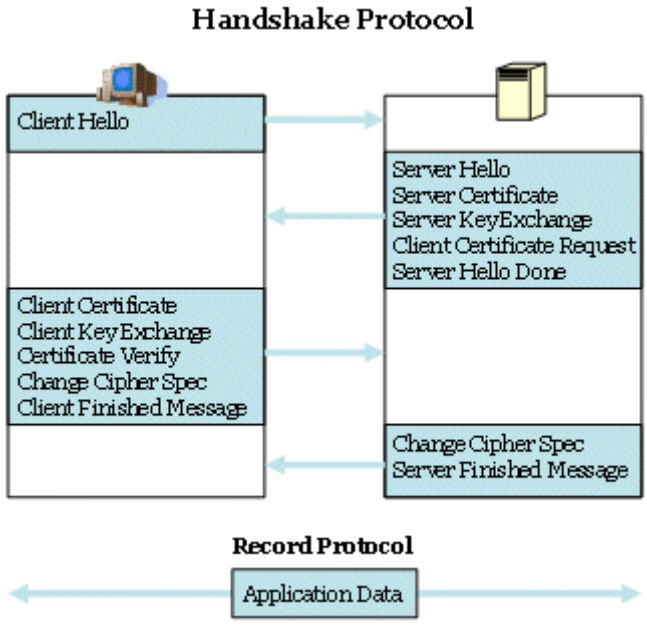


Figure 2 Handshake Protocol Messages

Initial Client Message to Server

Client Hello. The client initiates a session by sending a Client Hello message to the server. The Client Hello message contains:

- **Version Number**. The client sends the version number corresponding to the highest version it supports. Version 2 is used for SSL 2.0, version 3 for SSL 3.0, and version 3.1 for TLS. Although the IETF RFC for TLS is TLS version 1.0, the protocol uses 3.1 in the version field to indicate that it is a higher level (newer and with more functionality) than SSL 3.0.
- **Randomly Generated Data**. ClientRandom[32], the random value, is a 4-byte number that consists of the client's date and time plus a 28-byte randomly generated number that will ultimately be used with the

server random value to generate a master secret from which the encryption keys will be derived.

- **Session Identification (if any).** The sessionID is included to enable the client to resume a previous session. Resuming a previous session can be useful, because creating a new session requires processor-intensive public key operations that can be avoided by resuming an existing session with its established session keys. Previous session information, identified by the sessionID, is stored in the respective client and server session caches.
- **Cipher Suite**. The A list of cipher suites available on the client. An example of a cipher suite is TLS_RSA_WITH_DES_CBC_SHA, where TLS is the protocol version, RSA is the algorithm that will be used for the key exchange, DES_CBC is the encryption algorithm (using a 56-bit key in CBC mode), and SHA is the hash function.
- **Compression Algorithm**. The requested compression algorithm (none currently supported).

The following is an example of a Client Hello message:

```
ClientVersion 3,1
ClientRandom[32]
SessionID: None (new session)
Suggested Cipher Suites:
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
Suggested Compression Algorithm: NONE
```

Server Response to Client

Server Hello. The server responds with a Server Hello message. The Server Hello message includes:

- **Version Number**. The server sends the highest version number supported by both sides. This is the lower of: the highest version number the server supports and the version sent in the Client Hello message.
- **Randomly Generated Data.** ServerRandom[32], the Random Value, is a 4-byte number of the server's date and time plus a 28-byte randomly generated number that will be ultimately used with the client random value to generate a master secret from which the encryption keys will be derived
- **Session Identification (if any).** This can be one of three choices.
  - New session ID – The client did not indicate a session to resume so a new ID is generated. A new session ID is also generated when the client indicates a session to resume but the server can't or won't resume that session. This latter case also results in a new session ID.
  - Resumed Session ID– The id is the same as indicated in the client hello. The client indicated a session ID to resume and the server is willing to resume that session.
  - Null – this is a new session, but the server is not willing to resume it at a later time so no ID is returned.
- **Cipher Suite.** The server will choose the strongest cipher that both the client and server support. If there are no cipher suites that both parties support, the session is ended with a "handshake failure" alert.
- **Compression Algorithm**. Specifies the compression algorithm to use (none currently supported).

The following is an example of a Server Hello Message:

```
Version 3,1
ServerRandom[32]
SessionID: bd608869f0c629767ea7e3ebf7a63bdcffb0ef58b1b941e6b0c044acb6820a77
Use Cipher Suite:
TLS_RSA_WITH_3DES_EDE_CBC_SHA
Compression Algorithm: NONE
```

Server Certificate. The server sends its certificate to the client. The server certificate contains the server's public key. The client will use this key to authenticate the server and to encrypt the premaster secret.

The client also checks the name of the server in the certificate to verify that it matches the name the client used to connect. If the user typed www.contoso.com as the URL in the browser, the certificate should contain a subject name of www.contoso.com or *.contoso.com. Internet Explorer will warn the user if these names do not match.

Server Key Exchange. This is an optional step in which the server creates and sends a temporary key to the client. This key can be used by the client to encrypt the Client Key Exchange message later in the process. The step is only required when the public key algorithm does not provide the key material necessary to encrypt the Client Key Exchange message, such as when the server's certificate does not contain a public key.

Client Certificate Request. This is an optional step in which the server requests authentication of the client. This step might be used for Web sites (such as a banking Web site) where the server must confirm the identity of the client before providing sensitive information.

Server Hello Done. This message indicates that the server is finished and awaiting a response from the client.

Client Response to Server

Client Certificate. If the server sent a Client Certificate Request, the client sends its certificate to the server for client authentication. The client's certificate contains the client's public key.

Client Key Exchange. The client sends a Client Key Exchange message after computing the premaster secret using both random values. The premaster secret is encrypted by the public key from the server's certificate before being transmitted to the server. Both parties will compute the master secret locally and derive the session key from it.

If the server can decrypt this data and complete the protocol, the client is assured that the server has the correct private key. This step is crucial to prove the authenticity of the server. Only the server with the private key that matches the public key in the certificate can decrypt this data and continue the protocol negotiation.

This message will also include the protocol version. The server will verify that it matches the original value sent in the client hello message. This measure guards against *rollback attacks.* Rollback attacks work by manipulating the message in order to cause the server and the client to use a less secure, earlier version of the protocol.

Certificate Verify. This message is sent only if the client previously sent a Client Certificate message. The client is authenticated by using its private key to sign a hash of all the messages up to this point. The recipient verifies the signature using the public key of the signer, thus ensuring it was signed with the client's private key.

Change Cipher Spec. This message notifies the server that all messages that follow the Client Finished message will be encrypted using the keys and algorithms just negotiated.

Client Finished. This message is a hash of the entire conversation to provide further authentication of the client. This message is the first message that the record layer encrypts and hashes.

Server Final Response to Client

Change Cipher Spec Message. This message notifies the client that the server will begin encrypting messages with the keys just negotiated.

Server Finished Message. This message is a hash of the entire exchange to this point using the session key and the MAC secret. If the client is able to successfully decrypt this message and validate the contained hashes, it is assured that the SSL/TLS handshake was successful, and the keys computed on the client machine match those computed on the server.

The Alert Sub-Protocol

The alert sub-protocol is a component of the Handshake protocol that includes event-driven alert messages that can be sent from either party. Following an alert message the session is either ended or the recipient is given the choice of whether or not to end the session. The alerts are defined in the TLS specification in RFC 2246. Table 1 lists the alert messages and their descriptions.

Table 1 Event-Driven Alert Messages from the Alert Sub-Protocol

Alert Message	Fatal?	Description
unexpected_message	Yes	Inappropriate message
bad_record_mac	Yes	Incorrect MAC
decryption_failed	Yes	Unable to decrypt TLSCiphertext correctly
record_overflow	Yes	Record is more than 2^14+1024 bytes
handshake_failure	Yes	Unacceptable security parameters
bad_certificate	Yes	There is a problem with the Certificate
unsupported_certificate	No	Certificate is unsupported
certificate_revoked	No	Certificate has been revoked
certificate_expired	No	Certificate has expired
certificate_unknown	No	Certificate is unknown
illegal_parameter	Yes	Security parameters violated

unknown_ca	Yes	CA unknown
access_denied	Yes	Sender does not want to negotiate
decode_error	Yes	Unable to decode message
decrypt_error	Yes	Handshake cryptographic operation failed
export_restriction	Yes	Not in compliance with export regulations
protocol_version	Yes	Protocol version not supported by both parties
insufficient_security	Yes	Security requirements no met
internal_error	Yes	Error not related to protocol
user_canceled	Yes	Not related to protocol failure
no_renegotiation	No	Negotiation request rejected.

The Record Protocol

The record protocol receives the data from the application layer and:

- Fragments the data into blocks or reassembles fragmented data into its original structure.
- Numbers the sequence of data blocks in the message to protect against attacks that attempt to reorder data.
- Compresses or decompresses the data using the compression algorithm negotiated in the handshake protocol[6].
- Encrypts or decrypts the data using the encryption keys and cryptographic algorithm negotiated during the handshake protocol.
- Applies an HMAC (or a MAC for SSL 3.0) to outgoing data. Computes the HMAC and verifies that it is identical to the value transmitted in order to check data integrity when a message is received.

Once the record protocol has completed its operations on the data, it sends the data to the TCP/IP transport layer for transmission. If the data is incoming, it is sent to the appropriate process for reception.

SSL/TLS Scenarios

Many people think of SSL and TLS as protocols used with Web browsers for securely browsing the Internet. However, these are also general purpose protocols that can be used whenever authentication and data protection are necessary. The following examples depict a few uses of SSL/TLS today. This is not an exhaustive list. In fact the ability to access these protocols through the SSPI interface means that anyone take advantage of them for just about any application. Many applications are being modified to take advantage of the features of SSL/TLS.

Secure transaction with an e-commerce Web site. This is a typical use of SSL between a browser and a Web server. An example is an e-commerce shopping site where clients need to furnish their credit card numbers. The protocol would first confirm that the Web site's certificate was valid and then send the client's credit card information as cipher text. For this type of transaction, where the server's certificate is from a trusted source, only server-side authentication occurs. SSL/TLS would need to be enabled for the Web page, such as an order form, where the data transactions occur.

Authenticated client access to a secure Web site. Both the client and server need certificates from a mutually trusted CA. With Schannel, client certificates can be mapped on a one-to-one or many-to-one basis to their Windows Server 2003 user or computer accounts and can be managed by Active Directory Users and Computers. This is invisible to the users, who can be authenticated to a Website without needing to supply a password.

If you want to give several users access to confidential material, you can create a group, map the users' certificates to the group, and give the group permissions to the material.

In one-to-one mapping, the server has a copy of the client's certificate; whenever the client logs in, the server verifies that they are identical. This one-to-one mapping is typically used for private material, such as a banking site where only one individual has the right to view a personal account.

Remote Access. Schannel is used to provide authentication and data protection when users remotely log in to Windows-based systems or networks. Telecommuting is a common use for this technology. Users can more securely access their e-mail or enterprise applications from home or while traveling, reducing the risk of exposure of the information to anyone on the Internet.

SQL Access. Microsoft® SQL Server™ provides the ability for administrators to require authentication of the

client when connecting to the server running SQL Server. In addition, either the client or server can be configured to require encryption of the data transferred between them. Very sensitive information, such as financial or medical databases, can be protected to prevent unauthorized access and disclosure of information on the network.

E-mail. Exchange servers can use Schannel to protect data as it moves from server to server on the Intranet or Internet. Full end-to-end security might require the use of Secure/Multipurpose Internet Mail Extensions (S/MIME); however, the protection of data in a server-to-server exchange allows companies to use the Internet to securely transfer e-mail among divisions within the same company, subsidiaries and partners. This can be done regardless of whether S/MIME is used.

SSL and Firewalls

You must make some additional decisions if you need to conduct SSL/TLS transactions through a firewall. A firewall is a program that can exist in many different forms, but essentially functions as a barrier between your local area network (LAN) and the outside world. The SSL/TLS protocol interprets a computer on which a firewall is running as presenting a man-in-the-middle attack, which prevents the transaction from happening.

You can use one of two approaches to facilitate SSL/TLS transactions through a firewall:

- **Open the firewall to allow all traffic through a designated port.** The typical port for HTTP over SSL is 443. This port can be opened to allow traffic through to the destination Web server. Unfortunately, this means that the firewall can make security decisions based only on the apparent origin of the packet and its destination. The firewall cannot examine the encrypted data in the requests.
- **Configure the firewall or boundary system as a proxy server.** In this case, the boundary system is the destination for the SSL traffic from the client. The client will authenticate to the boundary system, which will then forward, or *proxy*, the requests to the internal system. The connection from the boundary system to the internal system might or might not be protected by using SSL. This presents an authentication problem because the proxy needs to transmit the authenticated identity of the original user to the internal system. It is not possible to use the certificate mapping features of Windows Server 2003 at the application server, because the authentication process that relies on the user's certificate takes place at the proxy.

Performance Considerations

Using the SSL protocol for encryption protects data traveling over the Internet, but it also imposes a performance penalty. SSL can slow down an application or Web site considerably. The most obvious performance degradation when using SSL/TLS results from the time required to establish a session and then encrypt and decrypt the data, all of which heavily use processor cycles.

There are ways to minimize this impact. This section discusses the performance issues related to the cryptography involved and ways to accelerate the performance of the public key operations in SSL/TLS by using hardware solutions. It also covers considerations for scaling Web sites that use SSL/TLS, and performance tuning options for Schannel in Windows Server 2003.

Asymmetric Encryption

The process of SSL/TLS key exchange can exact a performance penalty. Nearly all commercially available Web servers and clients implement and support RSA as the asymmetric key exchange algorithm. Although there are other key exchange algorithms supported by SSL/TLS, RSA is the most widely used algorithm.

Offload Hardware Advantages

The high computational cost of the RSA public-key encryption algorithm makes it attractive to use encryption offload hardware to accelerate processing. Using offload hardware offers two advantages over performing the RSA encryption in software:

- **Optimizes processing.** The general-purpose processors normally used in computers do most things very well. However, special processors optimized for RSA are able to perform specific mathematical operations much more efficiently.
- **Decreases system-wide load.** Offloading the large number of clock cycles required for RSA decryption to a specialized processor frees the computer's general purpose processor to do other things, such as serving Web pages to other clients, while the decryption completes.

Offload Devices

There are three types of offload devices: RSA offload; Hardware Security Modules (HSMs), and Internet Hardware Devices. The first two, RSA offload and HSM devices, are typically cards (such as PCI cards) that plug into the bus on the computer. They interact with the Windows Server 2003 operating system through the Crypto API (CAPI). Internet Hardware Devices are network devices.

- **RSA-offload only devices.** This type of device is designed to provide offloading of RSA operations from

the host CPU. These devices are typically cards that are installed with a minimum of effort and little to no configuration. They perform the RSA operation only, and offer no additional features, such as improved key security.

- **Hardware Security Modules (HSM).** These devices are primarily targeted to deployments where security of the private key is a priority. By default, private keys are stored in software. With HSMs, not only are the RSA operations offloaded, but private keys never leave the HSM. These devices are normally tamper-proof and meet varying levels of security certification.
- **Internet Hardware Devices.** This loosely-defined class of devices is generally targeted at solving various problems in deploying e-commerce and internet solutions. These devices all offload SSL operations from the Web Servers by terminating the SSL connections at the device and then proxy the request as standard HTTP. Many of these devices have additional features such as content caching and load-balancing.

Symmetric Encryption

Symmetric encryption is so efficient that offloading it to specialized hardware is unnecessary, and might even be counterproductive. For most symmetric algorithms, fewer clock cycles are required to process the data to be encrypted or decrypted on the main processor than to move it from main memory, through the I/O bus, onto the offload hardware, back through the bus and back into memory. Add in the necessary trip through the bus to the network adapter, and it becomes clear why this is commonly referred to as the triple-trip problem. Because all these trips would themselves require resources, and because most symmetric algorithms are optimized for computer processing, it is far more efficient to simply perform the operation on the processor.

Considerations for Scaling When Using SSL

Scaling out a Web site means adding servers and then sending client requests to these additional computers, thereby increasing the performance of the site. This is often done by using technologies such as software load balancing, hardware load balancing, and DNS round robin. The traditional methods for scaling out a Web site can still be utilized with SSL/TLS, but you must also deal with a specific requirement imposed by SSL/TLS.

Once an SSL/TLS session is established between a specific client and a specific server, only that server can encrypt and decrypt the requests and responses for that client. Consequently, during an SSL/TLS session a client cannot send an SSL response to a server different from the one with which the client has already negotiated. This would result in an error, and a new SSL/TLS session would have to be negotiated between the client and the new server.

Therefore, when scaling out an SSL/TLS site you must use a method that allows a client to maintain its session with the same SSL/TLS server. This requirement can be met when using Network Load Balancing in Windows Server 2003 by configuring affinity as Single or Class C. The Single option specifies that Network Load Balancing direct multiple requests from the same client IP address to the same cluster host. Class C affinity specifies that Network Load Balancing direct multiple requests from the same TCP/IP Class C address range to the same cluster host.

Most load balancing solutions provide similar functionality. For more information about Network Load Balancing, see The Network Load Balancing Technical Overview link on the Web Resources page.

For example, you might have an SSL/TLS site that runs on a multiprocessor server and can process only 100 transactions per second (Tx/Sec), but you require a processing rate of 300 Tx/sec. To meet this requirement, you could create a Network Load Balancing cluster using two additional servers of the same configuration, and configure the cluster for Single affinity or Class C affinity. In theory you would have increased the performance of your site to 300 Tx/sec. In fact, you would also need to take into account some slight differences in the configuration and possible overhead from using Network Load Balancing. However, it is clearly possible to scale out an SSL/TLS site so long as you observe the requirements imposed by SSL/TLS.

Another technique for increasing performance of an SSL/TLS site is *scaling up*. Scaling up refers to adding larger servers, adding additional processors, or adding special SSL/TLS accelerators to your environment. Scaling up will allow more transactions to be performed by the increase hardware capacity. There are no special considerations for SSL/TLS when scaling up a server.

Performance Tuning Parameters for SSL/TLS

The administrator of a Windows Server 2003–based system can tune several Schannel parameters to control session reconnects. These parameters can be used to increase performance or save memory in some application scenarios.

The Schannel Cache

SSL/TLS supports a reconnect operation that can be used to enable a client and server to resume a previously negotiated session. This is desirable in many cases because the SSL/TLS reconnect does not require the time needed for a full RSA handshake. But there are cases in which reconnects can be troublesome, such as during

performance testing or when the connection pattern of the site is that the same client never reconnects to the same Web server.

Reconnects are enabled by the Schannel cache, which keeps a list of previous SSL/TLS sessions that are established with the current credential handle. The sessions are referenced by the SSL/TLS session ID. Schannel currently maintains up to ten thousand cached sessions for a maximum of ten hours.

Schannel cache values are stored in the following registry subkey:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL

You can add or modify these values by using the registry editor Regedit.exe. Table 2 describes the entries and their values.

> **Caution** Do not edit the registry unless you have no alternative. The registry editor bypasses standard safeguards, allowing settings that can damage your system, or even require you to reinstall Windows. If you must edit the registry, back it up first and see the Resource Kit Registry Reference for Windows Server 2003.

Table 2 Schannel Registry Settings

Entry	Datatype	Description
MaximumCacheSize	REG_DWORD	The maximum number of SSL/TLS sessions to maintain in the cache. The default value is 10,000
ClientCacheTime	REG_DWORD	The time, in milliseconds, to expire each client side cache element. The default is 10 hours.
ServerCacheTime	REG_DWORD	The time, in milliseconds, to expire each server side cache element. The default is 10 hours.

Setting either MaximumCacheSize or ServerCacheTime to zero disables the server-side session cache and prevents reconnects. Increasing MaximumCacheSize or ServerCacheTime above the default values causes LSASS.EXE to consume additional memory. Each session cache element typically requires 2-4k bytes of memory.

Additional Reading

See the following resources for further information:

- RFC 2246: *The TLS Protocol Version 1.0*
- ITU-T Recommendation X.509, available from http://www.itu.int/home/index.html.

[1] See the web_client and web_server sample code in the SDK. These samples are located in Microsoft SDK\Samples\security\SSPI\SSL.

[2] At the time of publication of this paper, no compression algorithms have been standardized for use in either SSL or TLS.

[3] ITU-T X.509 is an International Telecommunication Union standard that defines the structure of a certificate.

[4] It is possible to have more than one original message that creates the same hash. An occurrence of this is called a collision. While collisions are theoretically possible, the likelihood of two meaningful messages having the same hash is extremely unlikely.

[5] Although the protocol allows compression, as of the date of publication of this paper, no compression algorithms have been standardized for TLS.

[6] As of the date of publication of this paper, no compression algorithms are defined or implemented.

Send feedback to Microsoft