NIST Special Publication 800-52

Guidelines for the Selection and Use of Transport Layer Security (TLS) Implementations

National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

C. Michael Chernick, Charles Edington III, Matthew J. Fanto, Rob Rosenthal

C O M P U T E R S E C U R I T Y

June 2005

NIST Special Publication 800-52

Guidelines for the Selection and Use of Transport Layer Security (TLS) Implementations

Recommendations of the National Institute of Standards and Technology

C. Michael Chernick, Charles Edington III, Matthew J. Fanto, Rob Rosenthal

C O M P U T E R S E C U R I T Y

Computer Security Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8930

June 2005

U.S. Department of Commerce

Donald L. Evans, Secretary

Technology Administration

Phillip J. Bond, Under Secretary of Commerce for Technology

National Institute of Standards and Technology

Arden L. Bement, Jr., Director

Table of Contents

Tables

Figures

Executive Summary

Office of Management and Budget (OMB) Circular A-130, *Management of Federal Information Resources,* requires managers of publicly accessible information repositories or dissemination systems that contain sensitive but unclassified data to ensure sensitive data is protected commensurate with the risk and magnitude of the harm that would result from the loss, misuse, or unauthorized access to or modification of such data. Given the nature of interconnected networks and the use of the Internet to share information, protection of this sensitive data can become difficult if proper mechanisms are not employed to protect the data. Transport layer security (TLS) provides such a mechanism to protect sensitive data during electronic dissemination across the Internet.

TLS is a protocol created to provide authentication, confidentiality and data integrity between two communicating applications. TLS is based on a precursor protocol called "The Secure Sockets Layer Version 3.0" (SSL 3.0) and is considered to be an improvement to SSL 3.0. SSL is specified in an expired Internet Draft working document of the Internet Engineering Task Force. Transport Layer Security Version 1 (TLS 1.0) specification is an Internet Request for Comments [RFC2246]. Each document specifies a similar protocol that provides security services over the Internet. While TLS 1.0 is based on SSL 3.0, and the differences are not dramatic; they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate. This Special Publication provides guidance to the selection and implementation of the TLS protocol while making effective use of Federal Information Processing Standards (FIPS) approved cryptographic algorithms, and suggests that TLS 1.0 configured with FIPS based cipher suites is the appropriate secure transport protocol.[1] Use of the recommendations provided in this Special Publication would promote:

- More consistent use of authentication, confidentiality and integrity mechanisms for the protection of information transport across the Internet;
- Consistent use of recommended cipher suites that encompass FIPS algorithms and open source technology; and
- Informed decisions by system administrators and managers in the integration of transport layer security implementations.

While these Guidelines are primarily designed for Federal users and system administrators to adequately protect sensitive but unclassified Federal Government data against serious threats on the Internet, they may also be used within closed network environments to segregate data. (The client-server model and security services discussed also apply in these situations). This Special Publication does not supercede any existing NIST publication and should be used in conjunction with existing policies and procedures.

[1] While SSL 3.0 is the most secure of the SSL protocol versions, it is not approved for use in the protection of Federal information because it relies in part on the use of cryptographic algorithms that are not FIPS-Approved. TLS when properly configured is approved for the protection of Federal information.

Reports on Information Security Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive, unclassified information in Federal computer systems. This Special Publication 800-series reports on ITL's research, guidance, and outreach efforts in computer security and its collaborative activities with industry, government, and academic organizations.

National Institute of Standards and Technology Special Publication 800-52
Natl. Inst. Stand. Technol. Spec. Publ. 800-52, 33 pages (June 2005)
CODEN: NSPUE2

Acknowledgments

The authors, Michael Chernick and Matt Fanto of NIST, and Charles Edington and Robert Rosenthal of Booz Allen Hamilton (BAH) would like to thank the many people who assisted with the development of this document. In particular we would like to acknowledge Marcus Sills of BAH who aided with researching the complexities of the TLS protocol and Elaine Barker of NIST who gave guidance on random number generation.

1 Introduction

Today, many World Wide Web browsers and server applications rely on secure SSL and TLS communications to protect sensitive data transmitted through the Internet. Many books such as [Rescorla01], [Comer00], and [Hall00] describe the Internet's client-server model and communication protocol design principles. None guide Federal users and system administrators to adequately protect sensitive but unclassified Federal Government data against the most serious threats on the World Wide Web – eavesdropping, data tampering and message forgery. Other books such as [Adams99] and [Housley01] as well as technical journal articles (e.g., [Polk03]) and NIST publications (e.g., [SP800-32]) describe how Public Key Infrastructure (PKI) can be used to protect information in the Internet.

It is assumed that the reader of these Guidelines is somewhat familiar with the ISO seven-layer model communications model (also known as the seven-layer stack) [7498], as well as the Internet and public key infrastructure concepts, including, for example, X.509 certificates. If not, the reader may refer to the references cited above in the first paragraph of this introduction for further explanations of background concepts that cannot be fully explained in these Guidelines.

These Guidelines briefly introduce computer communications architectural concepts. The Guidelines place the responsibility for communication security at the Transport layer of the OSI seven-layer communications stack, not within the application itself. Protection of sensitive but unclassified Government information can adequately be accomplished at this layer when appropriate protocol options are selected and used by clients and servers relying on transport layer security.

Unfortunately, security is not a single property possessed by a single protocol. Rather, security includes a complex set of related properties that together provide the required information assurance characteristics and information protection services. Security requirements are usually derived from a risk assessment to the threats or attacks an adversary is likely to mount against a system. The adversary is likely to take advantage of implementation vulnerabilities found in many system components including computer operating systems, application software systems, and the computer networks that interconnect them. These guidelines focus only on security within the network, and they focus directly on the small portion of the network communications stack that is referred to as the transport layer.

Usually, the best defense against telecommunications attacks is to deploy security services implemented with mechanisms specified in standards that are thoroughly vetted in the public domain and rigorously tested by third party laboratories, by vendors, and by users of commercial off-the-shelf products.

Three services that most often address network user security requirements are confidentiality, message integrity and authentication. A confidentiality service provides assurance that data is kept secret, preventing eavesdropping. A message integrity service provides confirmation that data modification is always detected thus preventing undetected deletion, addition, or modification of data. An authentication service provides assurance of the sender or receiver's identity, thereby preventing forgery.

2 Security in a Layered Communications Architecture

The layering of computer communication protocols into a protocol stack (as defined by the OSI Seven Layer Model [7498]) enables developers to design new communication systems using already defined protocols and specific communication requirements within each layer of the stack. Each layer of the transmitting system communicates with the corresponding layer on the receiving system(s). Within this communications stack, the functionality of each layer is, in theory, independent of each other layer. Placement of security services and implementation of the security mechanisms within the stack is specific to each individual layer of the stack. Since the OSI Seven Layer Model does not explicitly define where security services are to be placed, there has been some debate over the exact placement of security services and other implementation mechanisms. These debates continue as new standards evolve to meet the communication needs of users, local and wide area networking vendors, Internet Service Providers (ISPs), and World Wide Web application designers.

Today we know that data privacy, integrity, and authenticated message delivery are required in a client-server model, where a user's client browser accesses one or more web server's applications. In this model, the inter-network "fabric" of telephone lines, network routers, firewalls, and other network components is seldom under the control of the end user's client software or of the server's application software; and so, data protection against eavesdropping, tampering or message forgery must be placed within the client/server protocols higher in the stack above this inter-network fabric. This ensures that security services are controlled jointly by the client and server, and not by the "fabric".

In a typical Internet architecture, the protocols in this fabric are the Transmission Control and Internet Protocol (TCP/IP) stack plus the protocols below IP in the stack. Protocols below IP include local area network (LAN) protocols or other link protocols such as dial up, or directly connected modems, fiber optic links, or satellite links. The TCP/IP stack provides for the transmission of packets through complex arrangements of local, wide, or metropolitan area or globally connected sets of inter- or intra-networking technology.

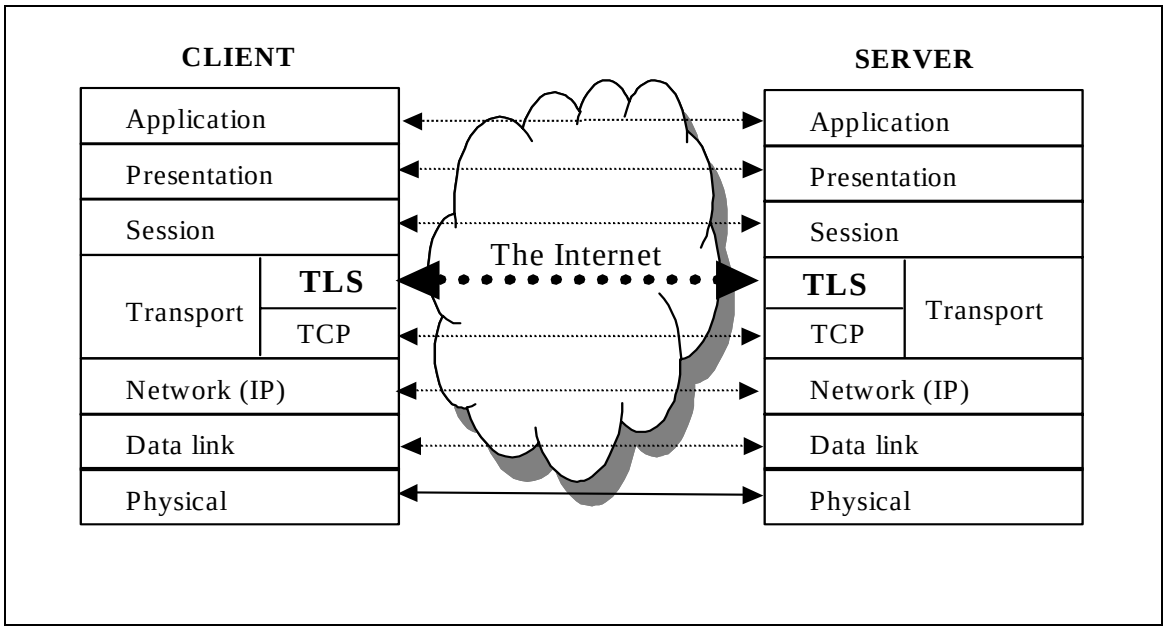

Figure 1: Client/Server Model[2]

Figure 1 depicts this architecture in the context of a client-server model. The arrow between the TLS protocol entities in Figure 1 emphasizes that the TLS entities make no assumptions about the security services provided by the interactions of the protocol entities lower in the stack. All of the security services needed to prevent eavesdropping, tampering or message forgery are provided by the TLS entities themselves. TLS does however rely on the communications functionality of the lower layer protocols to provide end-to-end reliable[3] delivery of data.

It is worth noting that TLS is not the only place in this architectural model where security services could be provisioned. As a general rule, placing services lower in the stack is advantageous because more higher layer entities can utilize the same functionality. However, placing security services lower in the model often requires sharing responsibility for security among many administrative organizations including the local, wide, or metropolitan area network and Internet Service Providers. Many feel that placing security in the transport layer, closer to, or part of, the client's browser and

[2] For simplicity and clarity firewalls and ISPs are not shown in this figure although in most deployed systems, firewalls and ISPs are used for connection to the Internet.

[3] Reliable delivery of data in this context has no security implication. Rather, reliable delivery of data implies that all messages presented to the sending TCP/IP stack are delivered in proper sequence by the receiving TCP/IP stack. These messages may be broken up into packets and fragmented or segmented as they are sent and routed through any arrangement of local, wide area or metropolitan networks. In general, as they are sent and routed through networks, the data are augmented with cyclical redundancy checks or forward error correction techniques to help ensure that the delivered messages are identical to the transmitted messages. Reliable delivery implies that the messages are properly reassembled and presented in correct order (proper sequence) to the peer protocol TLS entity.

server's application, is more appropriate for web-based applications rather than relying on or placing security services in lower levels.

2.1 Security in the Transport Layer

The Netscape Corporation[4] designed a protocol known as the Secure Sockets Layer (SSL) to meet security needs of client browsers and server applications. Version 1 of SSL was never released. Version 2 (SSL 2.0) was released in 1994 but had well-known security vulnerabilities. Version 3 (SSL 3.0) was released in 1995 to address these vulnerabilities. During this timeframe, Microsoft Corporation released a protocol known as Private Communications Technology (PCT), and later released a higher performance protocol known as the Secure Transport Layer Protocol (STLP). PCT and STLP never commanded the market share that SSL 2.0 and SSL 3.0 commanded. The Internet Engineering Task Force (IETF) (a technical working group responsible for developing Internet standards to ensure communications compatibility across different implementations), attempted to resolve, as best it could, security engineering and protocol incompatibility issues between the protocols. The IETF standards track Transport Layer Security Protocol Version 1.0 (TLS 1.0) emerged and was codified by the IETF as [RFC2246]. While TLS 1.0 is based on SSL 3.0, and the differences between them are not dramatic, they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate. However, TLS 1.0 does incorporate a mechanism by which a TLS 1.0 implementation can negotiate to use SSL 3.0 with requesting entities as if TLS were never proposed. However, because SSL 3.0 is not approved for use in the protection of Federal information, (Section 7.1 of [FIPS140Impl]), TLS must be properly configured to ensure that the negotiation and use of SSL 3.0 never occurs when Federal information is to be protected.

These Guidelines attempt to make clear the impact of selecting and using secure web transport protocols for use in protecting sensitive but unclassified U.S. Government information.

2.2 The Security Parts of Transport Layer Security

Both the TLS 1.0 and SSL 3.0 protocol specifications use cryptographic mechanisms to implement the security services that establish and maintain a secure TCP/IP connection. The secure connection prevents eavesdropping, tampering, or message forgery. Implementing data confidentiality with cryptography (encryption) prevents eavesdropping; generating a message authentication code with a secure hash function prevents undetected tampering; and, authenticating clients and servers with public key cryptography-based digital signatures prevents message forgery. In each case – preventing eavesdropping, tampering and forgery – a key or shared secret is required by the cryptographic mechanism. A pseudo-random number generator and a key establishment algorithm provide for the generation and sharing of these secrets.

[4] Commercial company names are used for historical reference purposes only. No product endorsement is intended or implied

The rows in Table 1 identify the key establishment, confidentiality, digital signature and hash mechanisms currently in use today in TLS 1.0 and SSL 3.0. For the purpose of these Guidelines, Table 1 identifies which key establishment, confidentiality, and signature algorithms and which hash functions are Federal Information Processing Standards (FIPS). These FIPS form the basis of the recommendations in these Guidelines.

Table 1: Mapping The Security Parts of TLS to Federal Standards

Mechanism	**SSL (3.0)**	**TLS 1.0**	**FIPS Reference**
Key Establishment	RSA DH-RSA DH-DSS DHE-RSA DHE-DSS DH-Anon Fortezza-KEA	RSA DH-RSA DH-DSS DHE-RSA DHE-DSS DH-Anon	
Confidentiality	IDEA-CBC RC4-128 3DES-EDE-CBC Fortezza-CBC	IDEA-CBC RC4-128 3DES-EDE-CBC Kerberos AES	FIPS 46-3, FIPS 81 FIPS 197
Signature	RSA DSA	RSA DSA EC*****	FIPS 186-2 FIPS 186-2 FIPS 186-2
Hash	MD5 SHA-1	MD5 SHA-1	 FIPS 180-2, FIPS 198

Note: DES and all of the "export" algorithms of small key sizes (RC4-40, RC2-CBC-40, DES-40, DHE-DSS-Export and DHE-RSA-Export) have been left out of this table as these are now deprecated.

*An Internet-Draft proposing cipher suites containing Elliptic Curve (EC) algorithms has been introduced in the IETF. See [ECCTLS].

Both TLS 1.0 and SSL 3.0 package one key establishment, confidentiality, signature and hash algorithm into a "cipher suite." Not all combinations from the table work together. Instead, TLS 1.0 and SSL 3.0 implementations contain carefully crafted cipher suites that are registered by the IETF (in the case of TLS 1.0) and the Netscape Corporation (in the case of SSL).[5]

[5] The transport layer security specification assigns a 16-bit (4 hexadecimal digit) number to the defined cipher suite. For example: Ephemeral Diffie-Hellman (DHE) **key agreement**, Digital Signature Algorithm (DSA) **signature**, Triple Data Encryption Standard (3DES) using Encryption-Decryption-Encryption (EDE) and Cipher Block Chaining (CBC) **confidentiality** and the Secure Hash Algorithm (SHA-1) **hash** is assigned the hexadecimal value {0x00, 0x13}. (Note that this suite is not frequently used.) Of special note is the TLS 1.0 requirement that, "In the absence of an application profile standard specifying otherwise, a TLS compliant application MUST implement the cipher suite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA.", which is represented by this example. However, the current draft of TLS 1.1 mandates TLS_RSA_WITH_3DES_EDE_CBC_SHA, and this suite is more commonly used.

A given implementation of the standard TLS 1.0 or SSL 3.0 protocol may implement one or more cipher suites from the registered set of suites. Finding a common cipher suite between a client and a server is accomplished through a built-in protocol "handshake negotiation" mechanism described in section 3, Negotiating Security Options, on page 13.

2.2.1 Key Establishment[6]

The following key establishment algorithms are used in various cipher suites.

RSA – The sender generates a random session key (pre-master secret) and encrypts it under the recipient's public key. The session key is a symmetric key.

DH (Diffie-Hellman) – The sender and receiver each have key pairs. To compute an agreed-upon session key, the sender combines his private key with the receiver's public key. The receiver combines his private key with the sender's public key.

There are three different types of DH communications, Static (also known as Fixed), Ephemeral, and Anonymous. These are described as follows[7]:

- **Fixed Diffie-Hellman:** This is a Diffie-Hellman key exchange in which the server's certificate[8] contains the Diffie-Hellman public parameters signed by the certificate authority (CA). That is, the public-key certificate contains the Diffie-Hellman public-key parameters. The client provides its Diffie-Hellman public key parameters either in a certificate, if client authentication is required, or in a key exchange message. This method results in a fixed secret key between two peers, based on the Diffie-Hellman calculation using the fixed public keys.

- **Ephemeral Diffie-Hellman:** This technique is used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, and signed using the sender's private RSA or DSS key. The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys. This option appears to be the most secure of the three Diffie-Hellman options because it results in a temporary, authenticated key.

---

[6] "Key Establishment" is the process of establishing a shared secret key used for encrypting data exchanged between client and server over a TLS connection. Key establishment is often called "key exchange". In some key establishment schemes (e.g., RSA), the client generates a random key and sends it to the server. In other schemes (e.g., Diffie-Hellman) the server generates some random data, sends the data to the client, the client generates additional random data, combines in with the server's random data, and the resulting key is sent to the server to be used as a secret key. This latter scheme is an example of a "key agreement" type of key establishment because the two sides together agree on a key.

[7] The Diffie-Hellman descriptions are used with the permission of Dr. William Stallings, and appeared in [Stallings98]

[8] For brevity, the term "certificate" is used in these guidelines to mean "X.509 certificate".

- **Anonymous Diffie-Hellman:** The base Diffie-Hellman algorithm is used, with no authentication. That is, each side sends its public Diffie-Hellman parameters to the other, with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie-Hellman exchanges with both parties.

Fortezza-KEA – KEA was the key agreement algorithm used by the Fortezza card supported by the Department of Defense, and KEA was originally classified. It exists in SSL 3.0, but the IETF standards committee did not include it in TLS 1.0.

The rules and protocols for generating and establishing keys, and the handling of those keys throughout their lifecycle, directly affects the level of security achieved. The security and reliability achieved depends on the strength of the key generation process and the protection afforded to those keys. Secret keys and the private key of a public key pair must be protected from disclosure, modification, substitution, and unauthorized deletion.

Some aspects of the key life cycle are addressed by the selection of appropriate cipher suites (i.e., the key establishment algorithm itself) or by the selection of validated modules[9] (e.g., primitives for key generation, storage, and destruction).

In deploying certificates for the support and use of TLS 1.0 and SSL 3.0 it is important to recognize that associated keys must be protected. Users must plan for and deal with many of these generally accepted cryptographic key life-cycle issues:

1) **Generation** of keys is accomplished with the aid of a pseudo-random number generator (PRNG). The protocol handshake sequence "ClientKeyExchange" creates a shared secret of random bits called the pre-master secret.

2) **Establishment** of keys occurs after the "ClientKeyExchange" message used in the handshake sequence creates the shared pre-master secret. This shared secret is expanded using a key derivation function in the client and the server. The key derivation function establishes the individual cryptographic keys that will be used for the various encryptions, authentication and the secure hash functions.[10]

3) **Storage** of keys and the shared secrets used to generate the keys is an issue in transport layer security implementations. Usually, the implementation relies on the capabilities of the operating system to protect this sensitive storage area.

4) **Crypto-periods** of both the session keys and certificate keys are not a transport layer security issue. The established session keys have a lifetime as long as the

[9] Cryptographic modules for Federal Government use must be validated in accordance with [FIPS140-1] or [FIPS140-2].

[10] The key derivation function uses the pre-master key, a label and a seed composed of shared random numbers to develop a master secret from which enough keying material is generated for all the algorithms defined in the cipher suite. This might include up to six keys – an encryption key, a MAC key (see section 2.2.5 for a description of Message Authentication Codes (MACs)) and an initialization vector for both the client and server.

session lasts.[11] Any keys used in the optional X.509 certificates that identify clients and servers have a crypto period that is determined by the policy of the certificate issuing authority.

5) **Recovery** of session keys is not a transport layer security issue. There is no requirement for a key recovery mechanism in the transport layer security protocol because the shared session key is ephemeral, a new session can be easily established with a new ephemeral key.[12]

6) **Destruction** of all of the state variables including the session keys occurs when the session ends. The protocol implementation relies on the operating system to insure that there is no reuse of storage areas, which may contain sensitive information.

2.2.2 Confidentiality

The following symmetric encryption algorithms are used in various cipher suites to provide confidentiality:

IDEA[13] – IDEA is a block cipher that operates on 64 bit plaintext blocks. The key is 128 bits long. The same algorithm is used for encryption and decryption. IDEA is not FIPS-approved.

RC4 – RC4 is a stream cipher that uses a variable length key of anywhere between 8 and 2048 bits long. RC4 is not FIPS-approved.

3DES-EDE – The Data Encryption Standard (DES) is the most widely used symmetric block cipher. It uses 64 bit blocks and a 56-bit key. Triple DES (also known as 3DES) super-encrypts by running the data through the DES algorithm 3 times with different keys. The first time it **E**ncrypts with key 1, the second time it **D**ecrypts with key 2 and the third time it **E**ncrypts again with key 3; hence the acronym 3DES-EDE. 3DES-EDE is FIPS-approved.

AES – The Advanced Encryption Standard is a FIPS-approved symmetric block cipher encryption algorithm that may be used by U.S. Government organizations (and others) to protect sensitive but unclassified information. AES uses 128, 192, or 256 bit keys, however cipher suites have only been defined for 128 and 256 bit keys to reduce the over proliferation of cipher suites. The block

---

[11] The message sequence space for SSL 3.0 and TLS 1.0 is 64 bits long; so, there is a theoretical limit of 2^{64} messages per session. If this limit is reached the connection must close as the sequence number cannot repeat or recycle. Furthermore, there is no mechanism to "rekey" the session. In effect this limits the crypto period of the session key.

[12] Servers and clients may (and often do) cache the master secret (but not the session key) to reduce the significant overhead in session resumption. After some reasonable timeout period, the master secret should be destroyed on both the server and the client.

[13] Cipher Block Chaining (CBC) is one of four DES modes of operation and can be used with all of the symmetric block ciphers listed here (i.e., IDEA, 3DES-EDE, and AES). CBC chains together the ciphertext of one block with the plaintext of the next block. Additional information on CBC can be found in [FIPS 46-3].

size in AES is 128 bits. The AES algorithm [FIPS197] is designed to replace DES and 3DES. AES is FIPS-approved.

Note that RC4 is currently the most commonly used confidentiality algorithm in SSL/TLS. However, RC4 is not FIPS-approved.

2.2.3 Signature

The following digital signature algorithms are used in various cipher suites:

RSA – To sign a message with the RSA algorithm the signatory computes a message digest or hash of the message and encrypts it with the private key. To verify an RSA signed message, the verifier decrypts the message digest with the signatory's public key and compares it with a locally computed hash of the original message. If the decrypted hash matches the locally calculated hash, the signature is valid. The use of RSA for digital signatures requires encryption and decryption.

DSA – To sign a message with the DSA algorithm, the signatory computes a signature using the SHA-1 hash algorithm and the signatory's private key. To verify a DSA signature, the verifier performs a computation using the message hash, the signatory's signature and the public key. The DSA verifier returns a yes or no rather than a decrypted hash as in the RSA signature. There is no encryption or decryption performed.

2.2.4 Hash

Hash algorithms used for cryptography (one-way hashes) have the property that it is computationally infeasible to find two messages that hash to the same value. When a message of any length is input to an algorithm, the result is an output called a message digest. The message digests range in length, depending on the algorithm used, however for use within Government environments four lengths have been specified; 160 bits, 256 bits, 384 bits, and 512 bits. Hash algorithms are identified as being secure if for a given algorithm, it is computationally infeasible that a message corresponding to a given message digest will be found, as well as finding two different messages producing the same message digest. Any change to a message will, with a very high probability, result in a very different message digest. This will cause a verification failure when the hash algorithm is used in conjunction with a digital signature algorithm or a keyed-hash message authentication algorithm.

The following hash algorithms are used in various cipher suites:

SHA–1: An algorithm for computing a condensed representation of a message or a data file. When a message of any length less than 2^{64} bits is input, SHA-1 produces a 160-bit output called a message digest. (The 2^{64} bit limit is caused by a 64-bit field size for a length descriptor used by SHA-1.)

MD5: When a message of any length is input, MD5 produces a 128-bit message digest. MD5 is not a FIPS-approved hash function, but it is in common use.

Note: Currently SHA-256, SHA-384, and SHA-512 are not featured in standard SSL or TLS cipher suites. It is expected that these variants of the secure hash algorithm will be added to TLS.

2.2.5 MAC

Message authentication (and integrity checking) is often achieved through the construction of a message authentication code (MAC), which is a small amount of additional data that is sent along with a message. To use a MAC, the sender and recipient must have a shared secret key known to no one else. Before sending a message, the sender computes the MAC as a function of the message and the key. Then the sender sends the message along with the computed MAC. The recipient computes a new MAC for the received message using the message itself, the secret key, and the same function as the sender. If the newly computed MAC is the same as the MAC received with the message, the recipient can be sure that the message has not been modified in transit, and that the sender knows the same secret key. That knowledge of the secret key assures the recipient of the sender's identity.

Cryptographic hash functions (with properties as described in 2.2.4 above) are often used as the function used to create MACs. In a simple scheme the secret key is simply prepended to the message, and the hash of this concatenation is used as the MAC. Message authentication codes generated using hash functions are known as HMACs and are used in TLS, SSL, and other common security mechanisms. For a detailed description of HMAC operation see [RFC2104] and [FIPS198].

A number of operations in the TLS protocol require an HMAC for authentication and integrity. Forging these MACs is infeasible without knowledge of the MAC secret key thus providing the necessary level of trust between communicating parties. Although HMACs can be used with a variety of different hash algorithms, TLS only specifies use of MD5 and SHA-1. To provide additional security for handshakes within the protocol, TLS uses both MD5 and SHA-1 in conjunction with each other to produce the HMAC.

3 Negotiating Security Options

Figure 2 provides details of the transport layer security protocol entities. The entities have been developed to allow control information to flow between the client and the server. Three types of control messages are exchanged: "Handshake", "ChangeCipherSpec, and "Alert". In addition, the entities exchange application data between the client and server protected by the security services provisioned by the negotiated cipher suite. These security services are negotiated and established with the Handshake.

The "Handshake" consists of a series of message exchanges between the client and the server. This series of messages is used to establish the TLS/SSL environment, including a set of negotiated security algorithms. The ChangeCipherSpec message is used to inform the other side to begin using the negotiated security services by changing to the cipher suite agreed to during the Handshake. All messages sent after ChangeCipherSpec message are encrypted using the just negotiated cipher suite.

The Alert provides a way to signal special security channel events and errors, as well as asynchronous events that may occur during a TLS/SSL session. In addition, an alert message is used to terminate a session.

Specific details of the Handshake, ChangeCipherSpec and Alert message exchanges are beyond the scope of these Guidelines; but understanding that the Handshake negotiates a cipher suite is important to understanding how to configure and use Transport Layer Security.

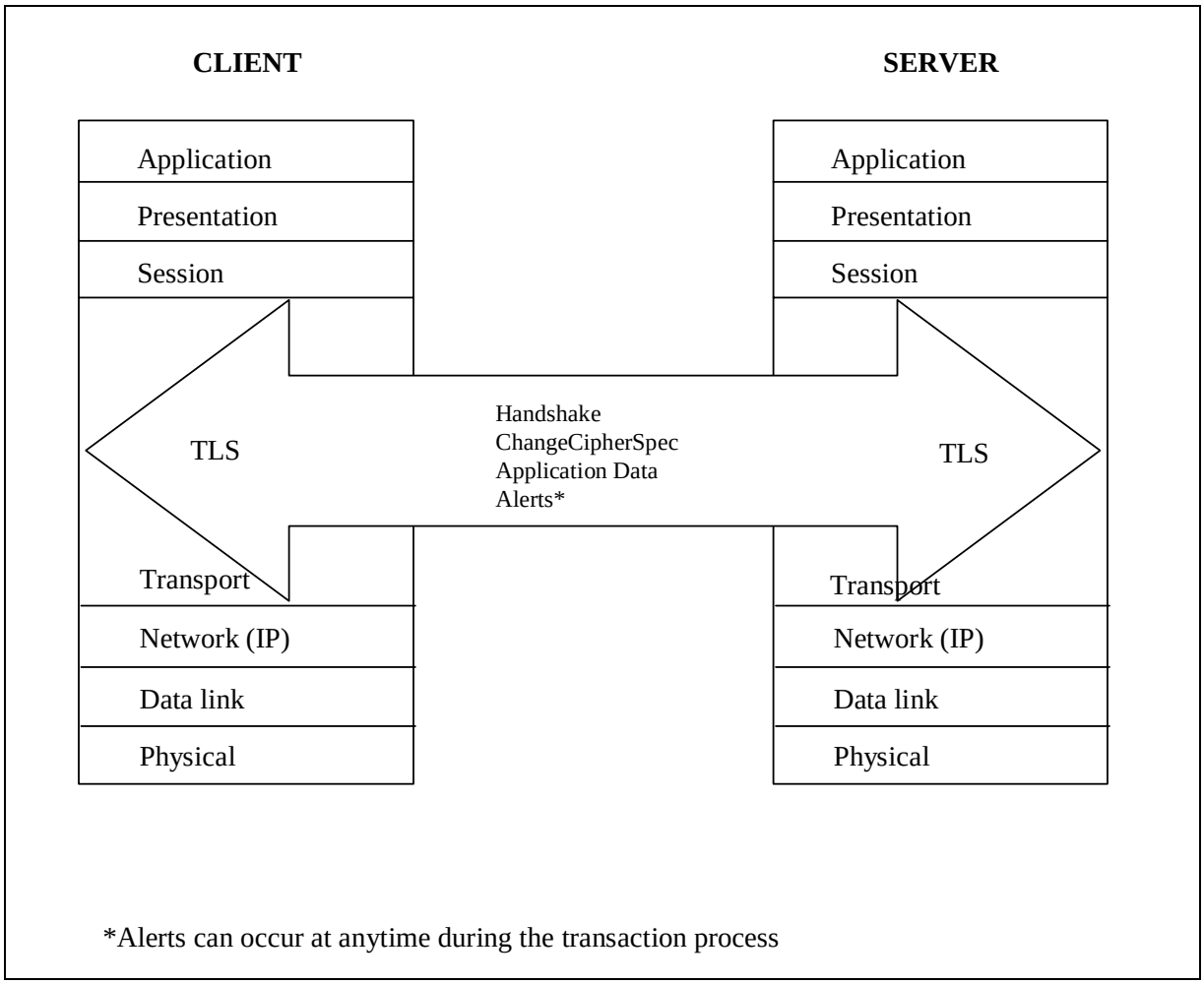

Figure 2: The Transport Layer Security Protocol Entity

The Transport Layer Security handshake protocol establishes a secure channel inside of a TCP/IP connection before passing any data from the application.

- The handshake protocol initializes both the client and server to use optional cryptographic capabilities by negotiating a cipher suite of algorithms and functions including key establishment, digital signature, confidentiality and integrity algorithms with their respective key sizes, and hash functions. This negotiation begins with the "ClientHello" message and continues with the "ServerHello" message.
- The handshake protocol may exchange public key digitally signed X.509 certificates[14] to optionally authenticate the server to the client and vice versa. In most cases, the server presents a certificate to the client, but the client does not present a certificate to the server. However, TLS and SSL allow for certificates to be presented by a server, by a client, by both, or by neither in negotiating a

[14] The use of X.509 certificates is fundamental to TLS/SSL, as well as other PKI-enabled services. For a comprehensive explanation of X.509 certificates see, for example, [Adams99] or [Housley01].

session. The important point to note is that presenting a valid X.509 certificate and proving possession of the private key authenticates the presenter to the recipient.

- Using the negotiated key establishment algorithm, the handshake protocol exchanges random data for developing keying material to be used by the cryptographic algorithms.

When all the security parameters are in place (i.e., after the Handshake), the "ChangeCipherSpec" message engages the negotiated cryptographic capabilities for application data exchanges.[15]

After the handshake protocol completes, application data may be exchanged between the client and server. This data is encrypted and integrity checked using the security services that were negotiated during the handshake.

Alerts are used to signal asynchronous/exceptional events during a TLS/SSL session. For example, an alert can be used to signal "decrypt_error" or "access_denied". Some alerts are used for warning, and others are considered "fatal" and can lead to immediate termination of the session. A "close_notify" alert is used to signal normal termination of a session. Like all other messages, after the handshake protocol is completed, alert messages are encrypted and optionally compressed.

3.1 The Cipher Suite

The negotiated algorithm identifiers are referred to collectively as the "Cipher Suite". Cipher suites are identified in human readable form using a mnemonic code. Each cipher suite includes a code for the protocol – either SSL or TLS. The next mnemonic identifies the key establishment algorithm and digital signature algorithm followed by the word "WITH" followed by the confidentiality algorithm followed by the hash function. An example is:

TLS_DHE_DSS_WITH_3DES-EDE-CBC_SHA

This cipher suite is for:

- A Transport Layer Security Version 1.0 protocol, TLS,
- The Ephemeral Diffie-Hellman key agreement, DHE,
- The Digital Signature Standard, DSS (which implies the Digital Signature Algorithm, DSA),
- The Triple Data Encryption Standard's Encrypt-Decrypt-Encrypt option in Cipher Block Chaining mode, 3DES-EDE-CBC, and
- The Secure Hash Algorithm, SHA-1 (used to compute a HMAC).

[15] Although not specifically related to security, the handshake also optionally negotiates a compression algorithm for the application data exchanges.

Both TLS (1.0) and SSL (3.0) specify registered index numbers for the cipher suites. When negotiating a cipher suite, the client (which always initiates TLS/SSL sessions) sends a handshake message with a list of cipher suite indexes it will accept. The server chooses from the list and sends a handshake message back indicating which cipher suite it will accept. Although the client orders the list with the "strongest" cipher suites listed first, the server may choose **ANY** of the cipher suites proposed by the client. Therefore there is **NO** guarantee that the negotiation will settle on the strongest suite in common. If no cipher suites are in common the connection is aborted.

When the negotiated options are complete, the client sends a "ChangeCipherSpec" and a "Finished" message. In response, the server also sends a "ChangeCipherSpec" and a "Finished" message to place the communication channel in a secure mode to protect transmission of the client's browser and server's application data.

3.2 Data Integrity of the Handshake

Data integrity is maintained throughout the handshake process and finally completed with the sending of the "Finished" message. A "Finished" message is always sent immediately after the "ChangeCipherSpec" message to verify the key exchange and authentication processes were successful. With successful exchanges of this message the client and server verify that the entire handshake has not been modified. This verification is possible because each side sends a hash of the concatenated handshake messages to the other side, which compares it to the same result computed locally. If the hash values differ, the handshake has been modified and the connection is aborted. If the hash values are the same, there is high assurance that the entire handshake has cryptographic integrity – nothing was modified, added or deleted.

4 Recommendations

This section presents criteria for developing specific recommendations when selecting, installing and using a transport layer security.

4.1 Selection Criteria

When implementing transport layer security mechanisms (usually web servers and browsers), there are several important aspects to consider such as whether the implementation:

- **Is standards-based –** The interaction between components in a transport layer security mechanism should be a well-defined communication protocol with no deviations. Additionally, FIPS-approved algorithms for authentication, encryption, and the generation of message digests should be used in all implementations.
- **Supports interoperability** – Any implementation should promote interoperability among components. The selection of a particular server solution should not prevent the use of any standards-based client or vise versa.
- **Includes evaluated products –** Key components of the implementation should be independently evaluated against known standards (e.g., cryptographic modules validated against FIPS 140-1 or 140-2).
- **Select important features –** The implementation should include those features that users consider most important to their operating environment.
- **Is Open Source –** The implementation should be an open source solution to help prevent being locked into a proprietary implementation that may not support interoperability or identified standards in the future.

4.2 Protocol Selection

Due to the vulnerabilities inherent in SSL 2.0, the only reasonable protocols to consider for deploying transport layer security are SSL 3.0 or TLS 1.0. However, using the criteria discussed in 4.1, TLS 1.0 is the only acceptable alternative. It is an IETF "standard" that incorporates FIPS-approved algorithms while SSL 3.0 is not standards-based, resulting in several competing incompatible SSL variants. Because SSL 3.0 uses some non-FIPS-approved cryptographic algorithms, it is not approved for use in the protection of Federal information [FIPS140Impl]. Likewise there exist incompatibilities between SSL 3.0 and TLS 1.0 implementations that prevent the integrated use of both TLS 1.0 and SSL 3.0 components, such as:

- SSL 3.0 allows a party to start sending application data as soon as that party's "Finished" message is sent. TLS requires a "wait" until the other "Finished" message is received.

- SSL 3.0 "ClientKeyExchange" contains two bytes representing the SSL version number. Some implementations use this value as a negotiation. Use of these two bytes as a negotiable value is incompatible with TLS 1.0.
- The key derivation from the pre-master secret is different. (SSL uses a key derivation model in which half the master secret is generated using only MD5, which is not a FIPS-approved hash algorithm.).
- SSL 3.0 omits the length bytes when RSA is used for sending the encrypted pre-master secret.
- Alert messages produce different results.

Although the selection of TLS may cause legacy problems with clients that are incapable of supporting TLS, almost all currently used clients (e.g., Internet Explorer, Netscape, Mozilla) support TLS.

4.3 Cipher Suite Selection

The last column in **Table 1: "Mapping the Security Parts of TLS to Federal Standards"** on page 7, lists the Federal Information Processing Standards that are applicable for use in various components of the negotiated cipher suites. Table 2 and Table 3 in Section 5 provide recommendations for the cipher suites that should be offered based, on the information presented below.

It is important to note that RFC 2246 specifies all TLS compliant applications implement the TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA cipher suite. This document does not supplant this requirement, but instead goes beyond RFC2246 and specifies only the following cipher suites may be used.

Key Establishment: For maximum security, RSA or DSA authentication with ephemeral Diffie-Hellman key agreement is recommended (e.g., AES might be TLS_DHE_RSA_WITH_AES-128_CBC_SHA and TLS_DHE_DSS_WITH_AES-256_CBC_SHA.).

These cipher suites provide perfect forward secrecy[16], and can support large keys for long-term confidentiality.

The Federal Government has not finalized a specification for FIPS-approved or NIST-Recommended algorithms for establishment of cryptographic keys. Note, however that a NIST Recommendation is currently under development. See [KeyMgmt03]. Upon completion, only recommended algorithms shall be used.

Confidentiality: Since TLS 1.0 is the security protocol of choice within the transport layer; only 3DES-EDE-CBC or AES should be offered during the handshake. Clients should never offer the deprecated 40 or 56-bit suites. While encryption is optional,

[16] Perfect forward secrecy is the condition in which the compromise of a session key or long-term private key after a given session does not cause the compromise of any earlier session key.

anonymous cipher suites are not allowed. For use in government agency to government agency communication, only FIPS-approved algorithms shall be used.[17]

Signature: The use of 1024 bit (key size) RSA/DSA server X.509 certificates is acceptable until the year 2010. If the client identity must be authenticated after the year 2010, key sizes larger than 1024 bits are needed.

Note that although RSA keys should have a key size of at least 1024 bits, many browsers and SSL/TLS servers have no mechanism to allow users to explicitly specify a minimum acceptable key size. However, by setting the symmetric key size to 128 bits (allowed by many browsers and servers) then an RSA key size of 1024 is often implicit in these browsers.

Hash: TLS offers two options for a cryptographic hash algorithm. These are SHA-1 and MD5. While SHA-1 is a FIPS-approved cryptographic hash algorithm, MD5 is not and thus should not be offered as a selection.

Note: The choice of a hash function within a cipher suite selection affects only the HMAC used to protect payload traffic. Other uses of hashes in key derivation or signatures are directly affected by cipher suite choice: RSA client authentication signature always uses both SHA-1 and MD5 while DSA client signatures only uses SHA-1.

[17] RC4 is acceptable for use on Government "Client" systems in very limited circumstances where secure information is to be transferred between Government systems and non-government servers, and 3DES or better (e.g., AES) is not supported by the server. For example, many vendor web sites providing supplies to the government support nothing stronger than RC4, and credit card information must be conveyed and secured to order supplies. In such cases risk is limited to exposure of government credit card information, and agencies may wish to take this risk to expedite ordering of supplies. RC4 should never be used on Government "Server" systems where government owned/generated data is to be made available in a secure manner to "client" systems.

5 Guidance

This section offers guidance to system administrators responsible for procuring, installing and maintaining implementations of transport layer security protocols.

Implementation Selection/Procurement – Among the responsibilities of any implementer or vendor of security functionality are:

- To provide quality random numbers for key generation,
- To protect the keying material and its storage,
- To properly implement and test key establishment, encryption, and signature algorithms and hash functions.

The procurement authority should ensure that the implementation meets a minimum set of universally accepted tests. Guidance for buying security products is provided in [SP800-23], Guidelines to Federal Organizations on Security Assurance and Acquisition/Use of Tested/Evaluated Products and [SP800-36], Guide to Selecting Information Technology Security Products.

Installation – Installation guidance usually involves following the vendor's general guidelines for configuration of the TLS or SSL protocol and following local (either to the server or to the client) configuration policy for selecting the optional protocol security services. For example, a client's local policy might state that server authentication is required. The system administrator would follow the vendor's prescribed methods for enabling client/server authentication.

The following security services should be configured and provided by the TLS implementation:

- Confidentiality,
- Data integrity,
- Peer entity authentication for clients and servers.

Appropriate cipher suites must also be selected and their priority ordered from strongest to weakest acceptable.

When peer entity authentication is selected, special attention should be paid to the public key infrastructure constructs used by TLS and SSL. Here, X.509 certificates are used. For example, for a client to be authenticated to a server, the authentication process requires that there must be a valid certification path that starts with one of the client's trust roots[18]. Part of the installation process is to ensure that all required trusted roots are present in the client and server implementations.

[18] A "trust root" (also known as a "trust anchor") is a X.509 certificate issued by (and signed by) a trusted authority. Before accepting a X.509 certificate (e.g., one presented by a TLS server) as valid, the user (known as a "relying party") must check to see that the certificate is signed by a trust root, or signed by an intermediate trusted authority whose certificate is signed by a trust root. There may be multiple intermediate trusted authorities, but the user must be able to find a chain of certificates that can traced back to a trust root before the user can rely upon a X.509 certificate. For a comprehensive explanation of X.509 certificates and trust see, for example, [Adams99] or [Housley01].

Note that for most common usage TLS and SSL are used to authenticate servers and not to authenticate clients. For merchandising implementations, for example, clients need to validate that they are dealing with an authenticated merchant before entering credit card information. The merchant only cares that a valid credit card is presented. (He does this by communicating with the credit card issuer. The details of that validation are beyond the scope of these guidelines.) The merchant does not need to authenticate the buyer.

Although outside of the installation process for the TLS implementation, another consideration is the adequate storage and protection of the client's and/or server's private key within a secure cryptographic module or token. This will prevent the masquerading of one party within the connection.

Maintenance – Once the server or client implementation is installed and operational, maintenance of the product generally follows local policies and operating procedures. For example, the site system administrator may be required to check for product updates and patches and install as needed. Within the local operating procedures, provisions need to be made for checking for and obtaining updated certificate revocation lists (CRLs) or using any on-line validation mechanisms available from the Certification Authority[19].

5.1 Considerations for Selecting TLS Client Implementations

Clients play a limited, but crucial role in the overall security posture. The client negotiates three parameters: the protocol version, the cipher suite, and the compression algorithm. These items are presented in the "ClientHello" message and form the basis for the server to negotiate the strongest possible security options.

The ClientHello message is the first message to be sent as the client establishes a TLS connection to the server. These messages allow the client to stay connected, re-establish an existing session, or to establish several independent secure sessions without repeating the full handshake procedure.

The client version field within the ClientHello message represents the protocol version that client supports. This field should contain the highest version number the client is prepared to support. For implementations that support TLS this value is: major=3, minor=1 (which represents 3.1, and hence TLS). All non-TLS implementations should use major=3, minor=0 for SSLv3. This designation does not limit the implementation to the identified protocol version. For example, if a client wishes to use only TLS, the client must connect to the server and is responsible for terminating the connection if the server selects any other protocol. Under no circumstances should a client use any protocol less than SSLv3[20]. For the most secure protection of data, only use clients that support TLS and that can disable all versions of SSL.

---

[19] Certificate authorities are trusted authorities that issue X.509 certificates to end users and to intermediate certificate authorities. When a certificate authority determines that a certificate has been compromised or should be considered invalid for further use, it adds the serial number of that certificate to its list of revoked certificates (i.e., to its CRL). Certificate authorities periodically distribute updated CRLs. Users (relying parties) should check the CRL before using an otherwise valid X.509 certificate to ensure that it has not been revoked.

[20] Only TLS can be used for the protection of Federal data. There may be instances where Federal users need access to non-Federal sites that do not support TLS. In these instances, Federal managers may allow the use of SSLv3 to transfer non-Federal data. However, the SSLv3 should be used in only in limited, low risk situations, and non-Federal sites should be encouraged to support TLS.

To ensure the security of the connection, client implementations should support the cipher suites presented in Table 2. This table presents the cipher suites listed in order of descending security strength. Unfortunately the order of these cipher suites is ignored by the server, which will select any cipher suite that it prefers from those offered by the client.

Table 2: Recommended Client Cipher Suites[21]

Cipher Suite	Authent-ication	Key Establishment	Encryption	Digest
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DSS	DHE	AES_256_CBC	SHA-1
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	RSA	DHE	AES_256_CBC	SHA-1
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	RSA	AES_256_CBC	SHA-1
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DSS	DH	AES_256_CBC	SHA-1
TLS_DH_RSA_WITH_AES_256_CBC_SHA	RSA	DH	AES_256_CBC	SHA-1
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DSS	DHE	AES_128_CBC	SHA-1
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	RSA	DHE	AES_128_CBC	SHA-1
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	RSA	AES_128_CBC	SHA-1
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DSS	DH	AES_128_CBC	SHA-1
TLS_DH_RSA_WITH_AES_128_CBC_SHA	RSA	DH	AES_128_CBC	SHA-1
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DSS	DHE	3DES_EDE_CBC	SHA-1
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DHE	3DES_EDE_CBC	SHA-1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	RSA	3DES_EDE_CBC	SHA-1
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DSS	DH	3DES_EDE_CBC	SHA-1
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DH	3DES_EDE_CBC	SHA-1
TLS_RSA_WITH_RC4_128_SHA[22]	RSA	RSA	RC4_128	SHA-1

At this time, compression options have not been defined for either TLS[23] or any version of SSL. However, OpenSSL and some proprietary implementations support private compression algorithms. Care should be given to ensure that these proprietary and/or private algorithms, if implemented, do not weaken the security posture of the protocol. Also, many implementations do not support compression, so compression may not be possible during a TLS/SSL session, even if desirable.

5.2 Server Considerations

Cipher Suites

Although the client may present the cipher suites that it prefers in order of descending preference, the server generally does not defer to the client's preferred cipher suite. The server may, at its choosing, select a common cipher suite that it prefers. The following

[21] TLS has standardized AES cipher suites, however, it is expected that these cipher suites are not yet widely supported by commercial products. As products begin to provide support for AES, client implementations should support AES cipher suites as the highest priority cipher suites.

[22] RC4 is not a FIPS-approved cryptographic algorithm. For this reason, cipher suites with RC4 should be offered only when communicating with non-government entities in limited, low risk situations for the transfer of non-Federal data when a FIPS-approved encryption algorithm is not supported. Normally this cipher suite should not be offered.

[23] An Internet Draft has been proposed to define compression methods for TLS [Hollenbeck04].

table (Table 3) represents the cipher suites that a TLS server implementation should support. This table presents the cipher suites in order of descending preference.

Table 3: Recommended Server Cipher Suites[24]

Cipher Suite	Auth	Key Establishment	Encryption	Digest
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DSS	DHE	AES_256_CBC	SHA-1
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	RSA	DHE	AES_256_CBC	SHA-1
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	RSA	AES_256_CBC	SHA-1
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DSS	DH	AES_256_CBC	SHA-1
TLS_DH_RSA_WITH_AES_256_CBC_SHA	RSA	DH	AES_256_CBC	SHA-1
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DSS	DHE	AES_128_CBC	SHA-1
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	RSA	DHE	AES_128_CBC	SHA-1
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	RSA	AES_128_CBC	SHA-1
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DSS	DH	AES_128_CBC	SHA-1
TLS_DH_RSA_WITH_AES_128_CBC_SHA	RSA	DH	AES_128_CBC	SHA-1
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DSS	DHE	AES_256_CBC	SHA-1
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DSS	DHE	3DES_EDE_CBC	SHA-1
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DHE	3DES_EDE_CBC	SHA-1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	RSA	3DES_EDE_CBC	SHA-1
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DSS	DH	3DES_EDE_CBC	SHA-1
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	RSA	DH	3DES_EDE_CBC	SHA-1

Note that all server certificates with RSA keys should have a key length of at least 1024 bits.

Client Authentication

Client authentication when required is accomplished during the handshake. The server initiates client authentication by requesting the client's certificate and providing guidance as to the types of certificates and algorithms the server will accept. The exchange is completed when the client responds with its certificate and a signed hash of the original handshake to prove possession of the corresponding private key.

Although the protocol allows the server to continue the connection using another authentication mechanism (e.g., username and password – not as strong but often used) if a client does not have a suitable certificate, for strong authentication or forgery prevention, server implementations should not allow the connection to be established. In the event that a client does not have a certificate or an acceptable certificate, the server should terminate that connection with a fatal "handshake failure" alert.

The TLS installations almost always require that servers use a certificate and their private key to authenticate themselves to clients. TLS client authentication is optional, and requires that the client have a suitable certificate, issued by a certification authority accepted by the server.

[24] TLS has standardized AES cipher suites [RFC3268], however, it is expected that these cipher suites are not yet widely supported by commercial products. As products begin to provide support for AES, server implementations should support AES cipher suites as the highest priority cipher suites.

In the most common use of TLS, only the server is authenticated through the TLS protocol itself. The client is assured that:

- the server has a certificate issued by a CA accepted by the client (as a practical matter this means a certificate issued by a CA with a self-signed root certificate that is in the in the client's trusted certificate store);
- the server controls the private key corresponding to the public key in the server's certificate, and;
- that the server's network address is consistent with the address in the server's certificate.

Browser clients typically display a "lock" symbol to inform the client that a secure, "https" session is in effect. The TLS session is frequently used primarily to protect a user password from eavesdroppers, and, in these cases, the authentication of the client to the server application depends entirely on the password, not the TLS protocol. Unless clients carefully manage the contents of the trusted certificate store and check the URI displayed in their browser's windows, it may be possible for a sophisticated "man-in-the-middle" attacker to successfully impersonate a web server and intercept protected data, including passwords.

Where strong cryptographic client authentication is required, the server should use the TLS protocol client authentication option to request a client certificate and use that certificate to cryptographically authenticate the client. The server can also provide the client with a list of the CAs it recognizes. In a successful client-authenticated TLS session both parties are assured that:

- the other party has a certificate issued by an acceptable CA (as a practical matter this means a certificate issued by a CA with a self-signed root certificate (possibly through intermediate CAs) that is in a trusted certificate store), and;
- the other party controls the private key corresponding to the public key in its certificate

In addition, the client knows that the server's network address is consistent with the address in the server's certificate.

When client authentication is used, both parties are strongly authenticated, and all data transferred in the TLS session is protected and bound to the authentication by symmetric keys generated in the authentication process. Unless clients carefully manage the contents of the trusted certificate store and check the URI displayed in their browser's windows, it may be possible for a sophisticated "man-in-the-middle" attacker to successfully impersonate a web server and intercept protected data. However, with client authenticated TLS, it is not possible for the attacker to learn the client's password or private key.[25]

[25] When TLS client authentication is used, there is no client password used, and the private key is never transmitted nor divulged.

Session Resumption

During the initial handshake between the client and server, the server generates a session id and passes this value to the client in the "ServerHello" message. The session id (along with the key material and cipher suite) is stored for later use after completion of the handshake. If the server is willing to resume a session at the request of a client (resubmission of "ClientHello"), the server responds with the original session id and cipher suite in the "ServerHello" message. In the event the server is unwilling to resume the session, the server generates and responds with a new session id.

Typical server implementations are agreeable to resuming a previous session. This is a secure mode of operation as the session keys (along with the pre-master secret and master secret) are known only to the client and server, and are coupled with the initial client authentication to provide the necessary security. If there is a requirement to authenticate each client as they initiate a connection session, the server should be configured to ignore requests to resume a session, and generate a new session id, which forces the entire handshake procedure (including client authentication) to proceed.

5.3 Generation of Random Numbers

Of particular concern to both the client and server is the generation of quality random numbers. Random numbers are used for the generation of keys, and for the generation of any random number that is needed to complete cryptographic exchanges. As such, it is important to strive for the following principles when generating random numbers:

- All possible outputs should occur with equal probability, and a series of outputs should appear to conform to a uniform distribution.
- Given a sequence of output bits, it should not be feasible to compute or predict any other (past or future) output bit.

Random numbers can be obtained using either a non-deterministic random bit generator (NRBG) or a deterministic (pseudorandom) random bit generator (DRBG). NRBGs and DRBGs produce bit strings from which random numbers can be determined. Both types of generators present implementation problems. DRBGs must be properly seeded with random data that should normally be kept secret, and the generator must provide the capability of generating a very large stream of bits without repeating. NRBGs rely on some unpredictable, physical source of randomness that is outside human control to produce random output. Typically, an NRBG may not produce random bits quickly enough. A simple solution to the problem of producing random numbers quickly is to use an NRBG to seed a DRBG. Care must be taken in the design and use of either type of generator to ensure that the requirements for randomness are met. Therefore, random numbers should be generated in modules that are validated under the Cryptographic Module Validation Program (CMVP)[26].

[26] The Cryptographic Module Validation Program (CMVP) is a joint venture of NIST and the Communications Security Establishment (CSE) of the Government of Canada. The CMVP validates commercial products for conformance to FIPS 140-1 ([FIPS140-1]) and FIPS 140-2 ([FIPS140-2]), allowing vendors to build to a common standard and utilize one common validation process. Additional information on the CMVP is available at Hhttp://cs-www.ncsl.nist.gov/cryptval/cmvp.htmH.

Note: This is an effort underway within the American National Standards Institute (ANSI) to develop a Random Number Generator standard. NIST plans are to use this as a basis for a Federal standard.

5.4 Operational Considerations

5.4.1 Implementation Considerations

Sections 5.1 and 5.2 of this document provide recommendations for the cipher suites that the clients and browsers should implement for transport layer security within the TLS protocol. System administrators need to fully understand the ramifications of selecting cipher suites and configuring applications to support only those cipher suites. Through current NIST research into products supporting TLS, it was determined that:

- The set of allowed cipher suites for most browsers includes, by default, RC4 for encryption with a 40 bit key,
- There was a limited choice of cipher suites for browsers and servers, and cipher suites with RC4 were typically chosen first,
- Most server implementations do not allow the server administrator to specify preference order. The only way to ensure that a server uses 3DES for encryption, was to configure the server to not implement cipher suites with RC4, and
- On many systems the selection of a cryptographic algorithm was system-wide and not application specific (e.g., disabling an algorithm for one application would disable that algorithm for all applications on Microsoft Windows systems using Microsoft's Cryptographic API).

Of equal importance is the need to specify the key lengths used in the cipher suites (for both clients and servers). Currently, most browsers do not allow for user specification of key lengths, but those utilizing OpenSSL libraries can select either 512 or 1024 bit RSA key sizes. The use of RSA/DSA server X.509 certificates with a minimum 1024 bit key size is acceptable until the year 2010. If the server identity must be authenticated after the year 2010, key sizes larger than 1024 bits are needed.

5.4.2 Additional Operational Concerns

The Hypertext Transfer Protocol (HTTP) is an extremely flexible protocol that allows for many uses and implementations. This flexibility, however, also introduces vulnerabilities that are not and cannot be mitigated solely by Transport Layer Security. Given the ease at which connections to a known server can be hijacked, it is incumbent upon the client to not only check all data received but also verify the pathway of the message and the message's integrity. This includes verifying the server's identity presented in the server's certificate at the time the connection is established.

Likewise, both the server and the client should not base authentication decisions solely upon the Transport Layer Security's mechanism for determining possession of the private

key that corresponds to the exchanged certificate. Rather, the decision should also consider whether or not the certificate is valid or has been revoked.

6. References

The following list of documents, publications, and organizations provide a wide variety of information on varying aspects of Transport Layer Security.

[Adams99] Adams, C. and Lloyd, S., *Understanding PKI: Concepts, Standard, and Deployment Considerations*, (Macmillan Technology Publishing, Indianapolis, IN, ISBN 1-57870-166-X, 1999).

[Comer00] Comer, D. E., *Internetworking with TCP/IP, Principles, Protocols, and Architectures*, Fourth Edition, (Prentice Hall, Upper Saddle River, NJ 07458, ISBN: 0-13- 018380-6, 2000).

[ECCTLS] Gupta, V. *ECC Cipher Suites for TLS*. Draft Internet Engineering Task Force, Request for Comment, October 2004. http://www.ietf.org/proceedings/04aug/I-D/draft-ietf-tls-ecc-06.txt

[Fanto2002] Fanto, M, SSL Web Server and Client Configuration, PKI Technical Working Group Presentation. http://csrc.nist.gov/pki/twg/y2002/presentations/twg-02-16.pdf

[FIPS46-3] FIPS 46-3, *Data Encryption Standard (DES)*[27], http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[FIPS81A] National Institute of Standards and Technology, *DES Modes of Operation*, Federal Information Processing Standard 81, 1980 December 2, http://csrc.nist.gov/publications/fips/fips81/fips81.htm

[FIPS81B] National Institute of Standards and Technology, *DES Modes of Operation Change Notice 2*, Federal Information Processing Standard 81, 1996 May31, http://csrc.nist.gov/publications/fips/fips81/fips81change2.pdf

[FIPS81C] National Institute of Standards and Technology, *DES Modes of Operation Change Notice 3*, Federal Information Processing Standard 81, http://csrc.nist.gov/publications/fips/fips81/fips81change3.pdf

[FIPS140-1] FIPS 140-1, *Security Requirements For Cryptographic Modules*, http://csrc.nist.gov/publications/fips/fips140-1/fips1401.pdf

[FIPS140-2] FIPS 140-2, *Security Requirements For Cryptographic Modules*, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[FIPS140Impl] National Institute of Standards and Technology, *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*, April 28, 2004, http://csrc.ncsl.nist.gov/cryptval/140-1/FIPS1402IG.pdf

[FIPS180-2] National Institute of Standards and Technology, *Secure Hash Standard (+ Change Notice to include SHA-224)*, Federal Information Processing Standards Publication 180-2, August 1 2002, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[27] FIPS 46-3 is in the process of being withdrawn and replaced by NIST Special Publication 800-67, currently available at Hhttp://csrc.nist.gov/publications/nistpubs/800-67/SP800-67.pdfH

[FIPS186-2] National Institute of Standards and Technology, *Digital Signature Standard*[28], Federal Information Processing Standard 186-2, 27 January 2000, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf.

[FIPS197] National Institute of Standards and Technology, *Advanced Encryption Standard (AES)*, Federal Information Processing Standard 197, November 26, 2001 http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[FIPS198] National Institute of Standards and Technology, *The Keyed-Hash Message Authentication Code (HMAC)*, Federal Information Processing Standard 198, 6 March 2002, http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf.

[Hall00] Hall, E. A., *Internet Core Protocols, The Definitive Guide*, (O'Reilly & Associates, ISBN: 1-56592-572-6, February 2000).

[Hollenbeck04] Hollenbeck, S., *Transport Layer Security Protocol Compression Methods*, Internet Engineering Task Force, Internet Draft, 16 January 2004. Transport Layer Security Protocol Compression Methods, http://www.ietf.org/internet-drafts/draft-ietf-tls-compression-07.txt (expires July 16, 2004). (Note: See http://www.ietf.org/html.charters/tls-charter.html for updated drafts and status.)

[Housley01] Housley, R. and Polk, T., *Planning for PKI, Best Practices Guide for Deploying Public Key Infrastructure*, (John Wiley & Sons, New York, NY, ISBN 0-471-39702-4, 2001).

[KeyMgmt03] National Institute of Standards and Technology's Computer Security Resource Center, *Key Management Information*, http://csrc.ncsl.nist.gov/CryptoToolkit/tkkeymgmt.html.

[PKCS1] RSA Laboratories, *PKCS #1 v2.1: RSA Cryptography Standard*, 14 June 2002.

[Polk03] Polk, W., Hastings, N., and Malani, A., *Public Key Infrastructures that Satisfy Security Goals*, IEEE Internet Computing, Volume 7, Number 4, July-August, 2003.

[Rescorla01] Rescorla, E., *SSL and TLS – Designing and Building Secure Systems*, (Addison- Wesley, Upper Saddle River NJ, 07458, ISBN 0-201-61598, March 2001).

[RFC2246] Dierks, T. and Allen, C., *The TLS Protocol Version 1.0*, Internet Engineering Task Force, Request for Comment 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC3268] Chown, P., *Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)*, Internet Engineering Task Force, Request for Comment 3268, June 2002, http://www.ietf.org/rfc/rfc3268.txt

[SP800-23] NIST Special Publication 800-23, *Guidelines to Federal Organizations on Security Assurance and Acquisition/Use of Tested/Evaluated Products*, August 2000, http://csrc.nist.gov/publications/nistpubs/800-23/sp800-23.pdf.

[28] FIPS 186-2 will be replaced by FIPS 186-3, which is currently out for comment.

[SP800-32] NIST Special Publication 800-32, *Introduction to Public Key Technology and the Federal PKI Infrastructure*, February 2001, http://csrc.nist.gov/publications/nistpubs/800-32/sp800-32.pdf

[SP800-36] NIST Special Publication 800-36, *Guide to Selecting Information Technology Security Products*, October 2003, http://csrc.nist.gov/publications/nistpubs/800-36/NIST-SP800-36.pdf.

[Stallings98] Stallings, W., *SSL: Foundation for Web Security*, Internet Protocol Journal, Volume1, Number 1, June 1998, http://www.cisco.com/en/US/about/ac123/ac147/archived_issues/ipj_1-1/ssl.htmlT

[X9.31] American National Standards Institute, *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (Appendix A.2.4, Generating Pseudorandom Numbers) X9.31*, 1998.

[7498] ISO/IEC 7498-1: 1994(E), ITU-T Rec. X.200 (1994 E), Information Processing Systems - OSI Reference Model - The Basic Model.