DRAFT (2nd) NIST Special Publication 800-52 Revision 2

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

Kerry McKay
David Cooper

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

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

DRAFT (2nd) NIST Special Publication 800-52 Revision 2

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

Kerry McKay
David Cooper
Computer Security Division
Information Technology Laboratory

October 2018

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Under Secretary of Commerce for Standards and Technology

Authority

This publication has been developed by NIST in accordance with its statutory responsibilities under the Federal Information Security Modernization Act (FISMA) of 2014, 44 U.S.C. § 3551 *et seq.*, Public Law (P.L.) 113-283. NIST is responsible for developing information security standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

National Institute of Standards and Technology Special Publication 800-52 Revision 2
Natl. Inst. Stand. Technol. Spec. Publ. 800-52 Rev. 2, 71 pages (October 2018)
CODEN: NSPUE2

Public comment period: *October 15, 2018* through *November 16, 2018*

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: sp80052-comments@nist.gov

All comments are subject to release under the Freedom of Information Act (FOIA).

Reports on Computer Systems 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 analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems. The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

Abstract

Transport Layer Security (TLS) provides mechanisms to protect data during electronic dissemination across the Internet. This Special Publication provides guidance to the selection and configuration of TLS protocol implementations while making effective use of Federal Information Processing Standards (FIPS) and NIST-recommended cryptographic algorithms. It requires that TLS 1.2 configured with FIPS-based cipher suites be supported by all government TLS servers and clients and requires support of TLS 1.3 by January 1, 2024. This Special Publication also provides guidance on certificates and TLS extensions that impact security.

Keywords

information security; network security; SSL; TLS; Transport Layer Security

Acknowledgements

The authors, Kerry McKay and David Cooper of the National Institute of Standards and Technology (NIST), would like to thank the many people who assisted with the development of this document. In particular, we would like to acknowledge Tim Polk of NIST and Santosh Chokhani of CygnaCom Solutions, who were co-authors on the first revision of this document. We would also like to acknowledge Matthew J. Fanto and C. Michael Chernick of NIST and Charles Edington III and Rob Rosenthal of Booz Allen and Hamilton who wrote the initial published version of this document.

Audience

This document assumes that the reader of these guidelines is familiar with TLS protocols and public-key infrastructure concepts, including, for example, X.509 certificates.

Note to Reviewers

The Triple Data Encryption Algorithm (TDEA), also known as 3DES, is no longer approved for use with TLS (see Department of Homeland Security Binding Operational Directive BOD-18-01, https://cyber.dhs.gov/assets/report/bod-18-01.pdf). The 64-bit block size does not provide adequate protection in applications such as TLS where large amounts of data are encrypted under the same key.

This draft also requires agencies to add support for TLS 1.3 by January 1, 2024. TLS 1.3 and 1.2 are intended to coexist, and support for both is encouraged after the TLS 1.3 adoption deadline.

Executive Summary

Office of Management and Budget (OMB) Circular A-130, *Managing Information as a Strategic Resource,* requires managers of public-facing information repositories or dissemination systems that contain sensitive but unclassified data to ensure that 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, the 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 protection 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 3.0 is specified in [31]. The Transport Layer Security version 1 (TLS 1.0) specification is an Internet Request for Comments, RFC 2246 [22]. Each document specifies a similar protocol that provides security services over the Internet. TLS 1.0 has been revised to version 1.1, as documented in RFC 4346 [23], and TLS 1.1 has been further revised to version 1.2, as documented in RFC 5246 [24]. In addition, some extensions have been defined to mitigate some of the known security vulnerabilities in implementations using TLS versions 1.0, 1.1, and 1.2. TLS 1.3, described in RFC 8446 [50], is a significant update to previous versions that includes protections against security concerns that arose in previous versions of TLS.

This Special Publication provides guidance to the selection and configuration of TLS protocol implementations while making effective use of NIST-approved cryptographic schemes and algorithms. In particular, it requires that TLS 1.2 be configured with cipher suites using NIST-approved schemes and algorithms as the minimum appropriate secure transport protocol and requires support for TLS 1.3 by January 1, 2024.[1] When interoperability with non-government systems is required, TLS 1.1 and TLS 1.0 may be supported. This Special Publication also identifies TLS extensions for which mandatory support must be provided and other recommended extensions.

The 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 transported across the Internet;
- Consistent use of the recommended cipher suites that encompass NIST-approved algorithms and open standards;
- Protection against known and anticipated attacks on the TLS protocol; and

[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 NIST-approved. TLS 1.2 is approved for the protection of Federal information when properly configured. TLS versions 1.1 and 1.0 are approved only when they are required for interoperability with non-government systems and are configured according to these guidelines.

- Informed decisions by system administrators and managers in the integration of TLS implementations.

While these guidelines are primarily designed for Federal users and system administrators to adequately protect sensitive but unclassified U.S. 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 supersedes NIST Special Publication 800-52 Revision 1. This Special Publication should be used in conjunction with existing policies and procedures.

Table of Contents

List of Appendices

1 Introduction

Transport Layer Security (TLS) protocols are used to secure communications in a wide variety of online transactions such as financial transactions (e.g., banking, trading stocks, e-commerce), healthcare transactions (e.g., viewing medical records or scheduling medical appointments), and social transactions (e.g., email or social networking). Any network service that handles sensitive or valuable data, whether it is personally identifiable information (PII), financial data, or login information, needs to adequately protect that data. TLS provides a protected channel for sending data between the server and the client. The client is often, but not always, a web browser. Memorandum M-15-13[2] requires that all publicly accessible Federal websites and web services only provide service through a secure connection.[3] The initiative to secure connections will enhance privacy and prevent modification of the data from government sites in transit.

TLS is a layered protocol that runs on top of a reliable transport protocol – typically the Transmission Control Protocol (TCP). Application protocols, such as the Hypertext Transfer Protocol (HTTP) and the Internet Message Access Protocol (IMAP), can run above TLS. TLS is application independent, and used to provide security to any two communicating applications that transmit data over a network via an application protocol.

1.1 History of TLS

The SSL protocol was designed by the Netscape Corporation to meet security needs of client and server applications. Version 1 of SSL was never released. SSL 2.0 was released in 1995, but had well-known security vulnerabilities, which were addressed by the 1996 release of SSL 3.0. During this timeframe, the 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 security engineering and protocol incompatibility issues between the protocols as best it could. The IETF standards track Transport Layer Security protocol Version 1.0 (TLS 1.0) emerged and was codified by the IETF as RFC 2246 [22]. 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.

TLS 1.1, specified in RFC 4346 [23], was developed to address weaknesses discovered in TLS 1.0, primarily in the areas of initialization vector selection and padding error processing. Initialization vectors were made explicit[4] to prevent a certain class of attacks on the Cipher Block Chaining (CBC) mode of operation used by TLS. The handling of padding errors was

[2] https://obamawhitehouse.archives.gov/sites/default/files/omb/memoranda/2015/m-15-13.pdf

[3] See https://https.cio.gov/ for more details on this initiative.

[4] The initialization vector (IV) must be sent; it cannot be derived from a state known by both parties, such as the previous message.

altered to treat a padding error as a bad message authentication code, rather than a decryption failure. In addition, the TLS 1.1 RFC acknowledges attacks on CBC mode that rely on the time to compute the message authentication code (MAC). The TLS 1.1 specification states that to defend against such attacks, an implementation must process records in the same manner regardless of whether padding errors exist. Further implementation considerations for CBC modes (which were not included in RFC 4346 [23]) are discussed in Section 3.3.2.

TLS 1.2, specified in RFC 5246 [24], made several cryptographic enhancements, particularly in the area of hash functions, with the ability to use or specify the SHA-2 family algorithms for hash, MAC, and Pseudorandom Function (PRF) computations. TLS 1.2 also adds authenticated encryption with associated data (AEAD) cipher suites.

TLS 1.3, specified in RFC 8446 [50], represents a significant change to TLS that aims to address threats that have arisen over the years. Among the changes are a new handshake protocol, a new key derivation process that uses the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [36], and the removal of cipher suites that use static RSA or DH key exchanges, the CBC mode of operation, or SHA-1. Many extensions defined for use with TLS 1.2 and below cannot be used with TLS 1.3.

1.2 Scope

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 of the threats or attacks that 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. Thus, in order to secure a system against a myriad of threats, security must be judiciously placed in the various systems and network layers.

These guidelines focus only on network security, and they focus directly on the small portion of the network communications stack that is referred to as the transport layer. Several other NIST publications address security requirements in the other parts of the system and network layers. Adherence to these guidelines only protects the data in transit. Other applicable NIST standards and guidelines should be used to ensure protection of systems and stored data.

These guidelines focus on the common use cases where clients and servers must interoperate with a wide variety of implementations, and authentication is performed using public-key certificates. To promote interoperability, implementations often support a wide array of cryptographic options. However, there are much more constrained TLS implementations where security is needed but broad interoperability is not required, and the cost of implementing unused features may be prohibitive. For example, minimal servers are often implemented in embedded controllers and network infrastructure devices such as routers, and then used with browsers to remotely configure and manage the devices. There are also cases where both the client and server for an application’s TLS connection are under the control of the same entity, and therefore allowing a variety of options for interoperability is not necessary. The use of an appropriate subset of the capabilities specified in these guidelines may be acceptable in such cases.

The scope is further limited to TLS when used in conjunction with TCP/IP. For example, Datagram TLS (DTLS), which operates over datagram protocols, is outside the scope of these guidelines. NIST may issue separate guidelines for DTLS at a later date.

1.2.1 Alternative Configurations

TLS may be used to secure the communications of a wide variety of applications in a diverse set of operating environments. As such, there is not a single configuration that will work well for all scenarios. These guidelines attempt to provide general-use recommendations. However, the needs of an agency or application may differ from general needs. **Deviations from these guidelines are acceptable, provided that agencies and system administrators assess and accept the risks associated with alternative configurations in terms of both security and interoperability.**

1.3 Document Conventions

Throughout this document, key words are used to identify requirements. The key words "**shall**," "**shall not**," "**should**," and "**should not**" are used. These words are a subset of the IETF Request for Comments (RFC) 2119 key words, and have been chosen based on convention in other normative documents [14]. In addition to the key words, the words "need," "can," and "may" are used in this document, but are not intended to be normative. The key words "NIST-approved" and "NIST-recommended" are used to indicate that a scheme or algorithm is described in a Federal Information Processing Standard (FIPS) or is recommended by NIST.

The recommendations in this document are grouped by server recommendations and client recommendations. Section 3 provides detailed guidance for the selection and configuration of TLS servers. Section 4 provides detailed guidance for the selection, configuration, and use of TLS clients.

2 TLS Overview

TLS exchanges records via the TLS record protocol. A TLS record contains several fields, including version information, application protocol data, and the higher-level protocol used to process the application data. TLS protects the application data by using a set of cryptographic algorithms to ensure the confidentiality, integrity, and authenticity of exchanged application data. TLS defines several protocols for connection management that sit on top of the record protocol, where each protocol has its own record type. These protocols, discussed in Section 2.1, are used to establish and change security parameters, and to communicate error and warning conditions to the server and client. Sections 2.2 through 2.6 describe the security services provided by the TLS protocol and how those security services are provisioned. Section 2.7 discusses key management.

2.1 TLS Subprotocols

There are three subprotocols in the TLS protocol that are used to control the session connection: the handshake, change cipher spec, and alert protocols. The TLS handshake protocol is used to negotiate the session parameters. The alert protocol is used to notify the other party of an error condition. The change cipher spec protocol is used in TLS 1.0, 1.1, and 1.2 to change the cryptographic parameters of a session. In addition, the client and the server exchange application data that is protected by the security services provisioned by the negotiated cipher suite. These security services are negotiated and established with the handshake.

The handshake protocol consists of a series of message exchanges between the client and the server. The handshake protocol initializes both the client and server to use cryptographic capabilities by negotiating a cipher suite of algorithms and functions, including key establishment, digital signature, confidentiality and integrity algorithms. Clients and servers can be configured so that one or more of the following security services are negotiated during the handshake: confidentiality, message integrity, authentication, and replay protection. A confidentiality service provides assurance that data is kept secret, preventing eavesdropping. A message integrity service provides confirmation that unauthorized data modification is detected, thus preventing undetected deletion, addition, or modification of data. An authentication service provides assurance of the sender or receiver's identity, thereby detecting forgery. Replay protection ensures that an unauthorized user does not capture and successfully replay previous data. In order to comply with these guidelines, both the client and the server must be configured for data confidentiality and integrity services.

The handshake protocol is used to optionally exchange X.509 public-key certificates[5] to authenticate the server and the client to each other.

The handshake protocol is responsible for establishing the session parameters. The client and server negotiate algorithms for authentication, confidentiality and integrity, as well as derive symmetric keys and establish other session parameters, such as extensions. The negotiated set of cryptographic algorithms is called the cipher suite.

[5] In these guidelines, the terms "certificate" and "public-key certificate" are used interchangeably.

Alerts are used to convey information about the session, such as errors or warnings. For example, an alert can be used to signal a decryption error (decrypt_error) or that access has been denied (access_denied). Some alerts are used for warnings, and others are considered fatal and lead to immediate termination of the session. A close_notify alert message 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 in TLS versions prior to TLS 1.3).

Details of the handshake, change cipher spec (in TLS versions prior to 1.3), and alert protocols are outside the scope of these guidelines; they are described in RFC 5246 [24] and RFC 8446 [50].

2.2 Shared Secret Negotiation

The client and server establish keying material during the TLS handshake protocol. The derivation of the premaster secret depends on the key exchange method that is agreed upon and the version of TLS used. For example, when Diffie-Hellman is used as the key-exchange algorithm in TLS 1.2 and earlier versions, the client and server send each other their parameters, which are used to compute the premaster secret. The premaster secret, along with random values exchanged by the client and server in the hello messages, is used in a pseudorandom function (PRF) to compute the master secret. In TLS 1.3, the master secret is derived by iteratively invoking an extract-then-expand function with previously derived secrets. The master secret is used to derive session keys, which are used by the negotiated security services to protect the data exchanged between the client and the server, thus providing a secure channel for the client and the server to communicate.

The establishment of these secrets is secure against eavesdroppers. When the TLS protocol is used in accordance with these guidelines, the application data, as well as the secrets, are not vulnerable to attackers who place themselves in the middle of the connection. The attacker cannot modify the handshake messages without being detected by the client and the server because the Finished message, which is exchanged after security parameter establishment, provides integrity protection to the entire exchange. In other words, an attacker cannot modify or downgrade the security of the connection by placing itself in the middle of the negotiation.

2.3 Confidentiality

Confidentiality is provided for a communication session by the negotiated encryption algorithm for the cipher suite and the encryption keys derived from the master secret and random values, one for encryption by the client (the client write key), and another for encryption by the server (the server write key). The sender of a message (client or server) encrypts the message using a derived encryption key; the receiver uses the same (independently derived) key to decrypt the message. Both the client and server know these keys, and decrypt the messages using the same key that was used for encryption. The encryption keys are derived from the shared master secret.

2.4 Integrity

The keyed MAC algorithm, specified by the negotiated cipher suite, provides message integrity. As with confidentiality, there is a different key for each direction of communication. The sender of a message (client or server) calculates the MAC for the message using the appropriate MAC key. When the receiver processes the message, it calculates its own version of the MAC using the MAC algorithm and sender's MAC key. The receiver verifies that the MAC that it calculates matches the MAC sent by the sender.

Two types of constructions are used for MAC algorithms in TLS. TLS versions 1.0, 1.1 and 1.2 support the use of the Keyed-Hash Message Authentication Code (HMAC) using the hash algorithm specified by the negotiated cipher suite. With HMAC, MACs for server-to-client messages are keyed by the server write MAC key, while MACs for client-to-server messages are keyed by the client write MAC key. These MAC keys are derived from the shared master secret.

TLS 1.2 added AEAD cipher modes of operation, such as Counter with CBC-MAC (CCM) [40] and Galois Counter Mode (GCM) [49, 53], as an alternative way of providing integrity and confidentiality. In AEAD modes, the sender uses its write key for both encryption and integrity protection. The client and server write MAC keys are not used. The recipient decrypts the message and verifies the integrity information using the sender's write key. In TLS 1.3, only AEAD symmetric algorithms are used for confidentiality and integrity.

2.5 Authentication

Server authentication is performed by the client using the server's public-key certificate, which the server presents during the handshake. The exact nature of the cryptographic operation for server authentication is dependent on the negotiated security parameters and extensions. In many cases, authentication is performed explicitly by verifying digital signatures using public keys that are present in certificates, and implicitly by the use of the server public key by the client during the establishment of the master secret. A successful Finished message implies that both parties calculated the same master secret and thus, the server must have known the private key corresponding to the public key in the server's certificate.

Client authentication is optional, and only occurs at the server's request. Client authentication is based on the client's public-key certificate. The exact nature of the cryptographic operation for client authentication depends on the negotiated cipher suite's key-exchange algorithm and the negotiated extensions. For example, when the client's public-key certificate contains an RSA public key, the client signs a portion of the handshake message using the private key corresponding to that public key, and the server verifies the signature using the public key to authenticate the client.

2.6 Anti-Replay

TLS provides inherent protection against replay attacks, except when 0-RTT data (optionally sent in the first flight of handshake messages) is sent in TLS 1.3.[6] The integrity-protected envelope of the message contains a monotonically increasing sequence number. Once the message integrity is verified, the sequence number of the current message is compared with the

[6] While TLS 1.3 does not inherently provide replay protection with 0-RTT data, the TLS 1.3 specification does recommend mechanisms to protect against replay attacks (see Section 8 of [50]).

sequence number of the previous message. The sequence number of the current message must be greater than the sequence number of the previous message in order to further process the message.

2.7 Key Management

The security of the server's private key is critical to the security of TLS. If the server's private key is weak or can be obtained by a third party, the third party can masquerade as the server to all clients. Similarly, if a third party can obtain a public-key certificate for a public key corresponding to its own private key in the name of a legitimate server from a certification authority (CA) trusted by the clients, the third party can masquerade as the server to the clients. Requirements and recommendations to mitigate these concerns are addressed later in these guidelines.

Similar threats exist for clients. If a client's private key is weak or can be obtained by a third party, the third party can masquerade as the client to a server. Similarly, if a third party can obtain a public-key certificate for a public key corresponding to his own private key in the name of a client from a CA trusted by the server, the third party can masquerade as that client to the server. Requirements and recommendations to mitigate these concerns are addressed later in these guidelines.

Since the random numbers generated by the client and server contribute to the randomness of the session keys, the client and server must be capable of generating random numbers with at least 112 bits of security[7] each. The various TLS session keys derived from these random values and other data are valid for the duration of the session. Because the session keys are only used to protect messages exchanged during an active TLS session, and are not used to protect any data at rest, there is no requirement for recovering TLS session keys. However, all versions of TLS provide mechanisms to store a key related to a session, which allow sessions to be resumed in the future. Keys for a resumed session are derived during an abbreviated handshake that uses the stored key as a form of authentication.

[7] See the SP 800-90 series for more information on random bit generators (https://csrc.nist.gov/projects/random-bit-generation)

3 Minimum Requirements for TLS Servers

This section provides a minimum set of requirements that a server must implement in order to meet these guidelines. Requirements are organized in the following sections: TLS protocol version support; server keys and certificates; cryptographic support; TLS extension support; client authentication; session resumption; compression methods; and operational considerations.

Specific requirements are stated as either implementation requirements or configuration requirements. Implementation requirements indicate that Federal agencies **shall not** procure TLS server implementations unless they include the required functionality, or can be augmented with additional commercial products to meet requirements. Configuration requirements indicate that TLS server administrators are required to verify that particular features are enabled or disabled, or in some cases, configured appropriately, if present.

3.1 Protocol Version Support

Servers that support government-only applications[8] **shall** be configured to use TLS 1.2, and **should** be configured to use TLS 1.3. These servers **should not** be configured to use TLS 1.1, and **shall not** use TLS 1.0, SSL 3.0, or SSL 2.0. TLS versions 1.2 and 1.3 are represented by major and minor number tuples (3, 3) and (3, 4), respectively, and may appear in that format during configuration.[9]

Servers that support citizen or business-facing applications (i.e., the client may not be part of a government IT system)[10] **shall** be configured to negotiate TLS 1.2, **should** be configured to negotiate TLS 1.3. The use of TLS versions 1.1 and 1.0 is generally discouraged, but these versions may be configured when necessary to enable interaction with citizens and businesses. See Appendix F for discussion on determining whether to support TLS 1.0 and TLS 1.1. These servers **shall not** allow the use of SSL 2.0 or SSL 3.0.

Agencies **shall** support TLS 1.3 by January 1, 2024. After this date, servers **shall** support TLS 1.3 for both government-only and citizen or business-facing applications. Note that TLS 1.3 and 1.2 are intended to coexist, and should both be enabled after the TLS 1.3 adoption deadline.

Some server implementations are known to implement version negotiation incorrectly. For example, there are TLS 1.0 servers that terminate the connection when the client offers a version newer than TLS 1.0. Servers that incorrectly implement TLS version negotiation **shall not** be

[8] A government-only application is an application where the intended users are exclusively government employees or contractors working on behalf of the government. This includes applications that are accessed on a government employee's bring-your-own-device (BYOD) system.

[9] Historically TLS 1.0 was assigned major and minor tuple (3,1) to align it as SSL 3.1. TLS 1.1 is represented by the major and minor tuple (3,2).

[10] For the purposes of this document, clients that reside on "bring your own device" (BYOD) systems, or privately-owned systems used to perform telework, are considered to be part of the government IT system, as they access services that are not available to the public.

used.

3.2 Server Keys and Certificates

The TLS server **shall** be configured with one or more public-key certificates and the associated private keys. TLS server implementations **should** support the use of multiple server certificates with their associated private keys to support algorithm and key size agility.

Several options for TLS server certificates meet the requirement for NIST-approved cryptography: an RSA signature certificate; an Elliptic Curve Digital Signature Algorithm (ECDSA) signature certificate; a Digital Signature Algorithm (DSA)[11] signature certificate; a Diffie-Hellman (DH) certificate; and an Elliptic Curve Diffie-Hellman (ECDH) certificate. Note that externally-accessible servers are expected to be configured with ECDSA or RSA certificates (see [67]). The other certificate types, and their associated cipher suites, are included in these guidelines for completeness and to cover edge cases.

At a minimum, TLS servers conforming to this specification **shall** be configured with an RSA signature certificate or an ECDSA signature certificate. If the server is configured with an ECDSA signature certificate, either curve P-256 or curve P-384 **should** be used for the public key in the certificate.[12]

TLS servers **shall** be configured with certificates issued by a CA. Furthermore, TLS server certificates **shall** be issued by a CA that publishes revocation information in Online Certificate Status Protocol (OCSP) [55] responses. The CA may additionally publish revocation information in a certificate revocation list (CRL) [18]. The source(s) for the revocation information **shall** be included in the CA-issued certificate in the appropriate extension to promote interoperability.

A TLS server that has been issued certificates by multiple CAs can select the appropriate certificate based on the client specified “Trusted CA Keys” TLS extension (see Section 3.4.2.7). A TLS server that has been issued certificates for multiple server names can select the appropriate certificate based on the client specified “Server Name” TLS extension (see Section 3.4.1.2). A TLS server certificate may also contain multiple names in the Subject Alternative Name extension in order to allow the use of multiple server names of the same name form (e.g., DNS name) or multiple server names of multiple name forms (e.g., DNS names, IP address, etc.).

Application processes for obtaining certificates differ and require different levels of proof when associating certificates to domains. An applicant can obtain a domain-validated (DV) certificate by proving control over a DNS domain. An Organization Validation (OV) certificate requires further vetting. An Extended Validation (EV) certificate has the most thorough identity vetting

[11] In the names for the TLS cipher suites, DSA is referred to as DSS (Digital Signature Standard), for historical reasons.

[12] The recommended elliptic curves now listed in FIPS 186-4 [62] will be moved to SP 800-186. Until SP 800-186 is published, the recommended elliptic curves should be taken from FIPS 186-4.

process. This recommendation does not provide guidance on which verification level to use.

Section 3.2.1 specifies a detailed profile for server certificates. Basic guidelines for RSA, ECDSA, DSA, DH, and ECDH certificates are provided. Section 3.2.2 specifies requirements for revocation checking. Section 3.5.4 specifies requirements for the "hints list."

3.2.1 Server Certificate Profile

The server certificate profile, described in this section, provides requirements and recommendations for the format of the server certificate. To comply with these guidelines, the TLS server certificate **shall** be an X.509 version 3 certificate; both the public key contained in the certificate and the signature **shall** provide at least 112 bits of security. Prior to TLS 1.2, the server Certificate message required that the signing algorithm for the certificate be the same as the algorithm for the certificate key (see Section 7.4.2 of [23]). If the server supports TLS versions prior to TLS 1.2, the certificate **should** be signed with an algorithm consistent with the public key:[13,14]

- Certificates containing RSA, ECDSA, or DSA public keys **should** be signed with those same signature algorithms, respectively;
- Certificates containing Diffie-Hellman public keys **should** be signed with DSA; and
- Certificates containing ECDH public keys **should** be signed with ECDSA.

The extended key usage extension limits how the keys in a certificate are used. There is a key purpose specifically for server authentication, and the server **should** be configured to allow its use. The use of the extended key usage extension will facilitate successful server authentication, as some clients may require the presence of an extended key usage extension. The use of the server DNS name in the Subject Alternative Name field ensures that any name constraints on the certification path will be properly enforced.

The server certificate profile is listed in Table 3-1. In the absence of agency-specific certificate profile requirements, this certificate profile **should** be used for the server certificate.

Table 3-1: TLS Server Certificate Profile

Field	Critical	Value	Description
Version	N/A	2	Version 3
Serial Number	N/A	Unique positive integer	Must be unique

[13] This recommendation is an artifact of requirements in TLS 1.0 and 1.1.

[14] Algorithm-dependent guidelines exist for the generation of public and private key pairs. For guidance on the generation of DH and ECDH key pairs, see SP 800-56A [7]. For guidance regarding the generation of RSA, DSA and ECDSA key pairs, see [62].

Field	Critical	Value	Description
Issuer Signature Algorithm	N/A	*Values by CA key type:*	
		sha256WithRSAEncryption {1 2 840 113549 1 1 11}, or stronger	CA with RSA key
		id-RSASSA-PSS {1 2 840 113549 1 1 10 }	CA with RSA key
		ecdsa-with-SHA256 {1 2 840 10045 4 3 2}, or stronger	CA with elliptic curve key
		id-dsa-with-sha256 {2 16 840 1 101 3 4 3 2}, or stronger	CA with DSA key
Issuer Distinguished Name (DN)	N/A	Unique X.500 issuing CA DN	A single value **shall** be encoded in each Relative Distinguished Name (RDN). All attributes that are of DirectoryString type **shall** be encoded as a PrintableString.
Validity Period	N/A	3 years or less	Dates through 2049 expressed in UTCTime
Subject Distinguished Name	N/A	Unique X.500 subject DN per agency requirements	A single value **shall** be encoded in each RDN. All attributes that are of DirectoryString type **shall** be encoded as a PrintableString. If present, the CN attribute **shall** be of the form: CN={host IP address | host DNS name}
Field	**Critical**	**Value**	**Description**
Subject Public Key Information	N/A	*Values by certificate type:*	
		rsaEncryption {1 2 840 113549 1 1 1}	RSA signature certificate 2048-bit RSA key modulus, or other approved lengths as defined in [62] and [5] Parameters: NULL
		ecPublicKey {1 2 840 10045 2 1}	ECDSA signature certificate or ECDH certificate Parameters: namedCurve OID for named curve specified in SP 800-186.[15] The curve **should** be P-256 or P-384 SubjectPublic Key: Uncompressed EC Point.
		id-dsa {1 2 840 10040 4 1}	DSA signature certificate Parameters: p, q, g (2048-bit large prime, i.e., p)
		dhpublicnumber {1 2 840 10046 2 1}	DH certificate Parameters: p, g, q (2048-bit large prime, i.e., p)
Issuer's Signature	N/A	Same value as in Issuer Signature Algorithm	
Extensions			

[15] The recommended elliptic curves now listed in FIPS 186-4 [62] will be moved to SP 800-186. Until SP 800-186 is published, the recommended elliptic curves should be taken from FIPS 186-4.

Field	Critical	Value	Description
Authority Key Identifier	No	Octet String	Same as subject key identifier in issuing CA certificate Prohibited: Issuer DN, Serial Number tuple
Subject Key Identifier	No	Octet String	Same as in PKCS-10 request or calculated by the issuing CA
Key Usage	Yes	*Values by certificate type:*	
		digitalSignature	RSA signature certificate, ECDSA signature certificate, or DSA signature certificate
		keyAgreement	ECDH certificate, DH certificate
Extended Key Usage	No	id-kp-serverAuth {1 3 6 1 5 5 7 3 1}	Required
		id-kp-clientAuth {1 3 6 1 5 5 7 3 2}	Optional
			Prohibited: anyExtendedKeyUsage; all others unless consistent with key usage extension
Certificate Policies	No		Optional
Subject Alternative Name	No	DNS host name, or IP address if there is no DNS name assigned	Required. Multiple SANs are permitted, e.g., for load balanced environments.
Authority Information Access	No	id-ad-caIssuers	Required. Access method entry contains HTTP URL for certificates issued to issuing CA
		id-ad-ocsp	Required. Access method entry contains HTTP URL for the issuing CA OCSP responder
CRL Distribution Points	No	See comments	Optional. HTTP value in distributionPoint field pointing to a full and complete CRL. Prohibited: reasons and cRLIssuer fields, and nameRelativetoCRLIssuer CHOICE
Signed Certificate Timestamps List	No	See comments	Optional. This extension contains a sequence of Signed Certificate Timestamps, which provide evidence that the certificate has been submitted to Certificate Transparency logs.
TLS feature	No	status_request(5)	Optional. This extension (sometimes referred to as the “must staple” extension) may be present to indicate to clients that the server supports OCSP stapling and will provide a stapled OCSP response when one is requested.

3.2.2 Obtaining Revocation Status Information for the Client Certificate

The server **shall** perform revocation checking of the client certificate when client authentication is used. Revocation information **shall** be obtained by the server from one or more of the following locations:

1. Certificate Revocation List (CRL) or OCSP [55] response in the server’s local store;
2. OCSP response from a locally configured OCSP responder;

3. OCSP response from the OCSP responder location identified in the OCSP field in the Authority Information Access extension in the client certificate; or

4. CRL from the CRL Distribution Points extension in the client certificate.

When the local store does not have the current or a cogent[16] CRL or OCSP response, and the OCSP responder and the CRL distribution point are unavailable or inaccessible at the time of TLS session establishment, the server will either deny the connection or accept a potentially revoked or compromised certificate. The decision to accept or reject a certificate in this situation **should** be made according to agency policy.

3.2.3 Server Public-Key Certificate Assurance

The policies, procedures, and security controls under which a public-key certificate is issued by a CA are documented in a certificate policy. The use of a certificate policy that is designed with the secure operation of PKI in mind and adherence to the stipulated certificate policy mitigates the threat that the issuing CA can be compromised or that the registration system, persons or process can be compromised to obtain an unauthorized certificate in the name of a legitimate entity, and thus compromise the clients. With this in mind, the CA Browser Forum, a private-sector organization, has carried out some efforts in this area by writing requirements for issuing certificates from publicly trusted CAs in order for those CAs and their trust anchor to remain in browser trust stores [15]. Under another effort, the CA Browser Forum has written guidelines for issuing Extended Validation Certificates [16].

Several concepts are under development that further mitigate the risks associated with the compromise of a CA or X.509 certificate registration system, process or personnel. These include the Certificate Transparency project (see Section 3.4.2.11) and other emerging concepts, which are discussed in Appendix E.

The policy under which a certificate has been issued may optionally be represented in the certificate using the certificatePolicies extension, specified in [18] and updated in [70]. When used, one or more certificate policy object identifiers (OID) are asserted in this extension, with each OID representing a specific certificate policy. Many TLS clients (e.g., browsers), however, do not offer the ability to accept or reject certificates based on the policies under which they were issued. Therefore, it is generally necessary for TLS server certificates to be issued by CAs that only issue certificates in accordance with a certificate policy that specifies adequate security controls.

When an agency is obtaining a certificate for a TLS server for which all the clients are under the agency's control, the agency may issue the certificate from its own CA if it can configure the clients to trust that CA. In other cases, the agency should obtain a certificate from a publicly-trusted CA (a CA that clients that will be connecting to the server have already been configured

[16] A CRL is considered "cogent" when the "CRL Scope" [18] is appropriate for the certificate in question.

to trust).

3.3 Cryptographic Support

Cryptographic support in TLS is provided through the use of various cipher suites. A cipher suite specifies a collection of algorithms for key exchange (in TLS 1.2 and earlier only),[17] and for providing confidentiality and integrity services to application data. The cipher suite negotiation occurs during the TLS handshake protocol. The client presents cipher suites that it supports to the server, and the server selects one of them to secure the session data.

In addition to the selection of appropriate cipher suites, system administrators may also have additional considerations specific to the implementation of the cryptographic algorithms, as well as cryptographic module validation requirements. Acceptable cipher suites are listed in Section 3.3.1, grouped by certificate type and protocol version. Implementation considerations are discussed in Section 3.3.2, and recommendations regarding cryptographic module validation are described in Section 3.3.3.

3.3.1 Cipher Suites

Cipher suites specify the cryptographic algorithms that will be used for a session. Cipher suites in TLS 1.0 through TLS 1.2 have the form:

TLS_*KeyExchangeAlg*_WITH_*EncryptionAlg_MessageAuthenticationAlg*

For example, the cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA uses ephemeral ECDH key establishment, with parameters signed using RSA, confidentiality is provided by AES-128 in cipher block chaining mode, and message authentication is performed using HMAC_SHA.[18] For further information on cipher suite interpretation, see Appendix B.

Cipher suites are defined differently in TLS 1.3. These cipher suites do not specify the key exchange algorithm, and have the form:

TLS_*AEAD_HASH*

For example, the cipher suite TLS_AES_128_GCM_SHA256 uses AES-128 in Galois Counter Mode for confidentiality and message authentication, and uses SHA-256 for the PRF. TLS 1.3 cipher suites cannot be used for TLS 1.2 connections, and TLS 1.2 cipher suites cannot be negotiated with TLS 1.3.

When negotiating a cipher suite, the client sends a handshake message with a list of cipher suites 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 may order the list with what it considers to be the strongest cipher suites listed first, the server may ignore the preference order and choose

---

[17] In TLS 1.3 the key exchange algorithm is specified solely in extensions (see Sections 3.4.2.3 and 3.4.2.10).

[18] SHA indicates the use of the SHA-1 hash algorithm.

any of the cipher suites proposed by the client. The server may have its own cipher suite preference order, and it may be different from the client's. Therefore, there is *no* guarantee that the negotiation will settle on the strongest common suite. If no cipher suites are common to the client and server, the connection is aborted.

The server **shall** be configured to only use cipher suites that are composed entirely of NIST-approved algorithms (i.e., [6, 7, 9, 25-27, 61-63, 65]). A complete list of acceptable cipher suites for general use is provided in this section, grouped by certificate type and TLS protocol version. The Internet Assigned Numbers Authority (IANA) value for each cipher suite is given after its text description, in parentheses.[19]

In some situations, such as closed environments, it may be appropriate to use pre-shared keys. Pre-shared keys are symmetric keys that are already in place prior to the initiation of a TLS session, which are used in the derivation of the premaster secret. For cipher suites that are acceptable in pre-shared key environments, see Appendix C.

NIST is deprecating the use of RSA key transport as used in TLS. Some applications or environments may require the use of RSA key transport during a transition period. Acceptable cipher suites for use in this situation are located in Appendix D.

The following cipher suite listings are grouped by certificate type and TLS protocol version. The cipher suites in these lists include the cipher suites that contain NIST-approved cryptographic algorithms. Cipher suites that do not appear in this section, Appendix C, or Appendix D **shall not** be used.

Cipher suites using ephemeral DH and ephemeral ECDH (i.e., those with DHE or ECDHE in the second mnemonic) provide perfect forward secrecy.[20] When ephemeral keys are used to establish the master secret, each ephemeral key-pair (i.e., the server ephemeral key-pair and the client ephemeral key-pair) **shall** have at least 112 bits of security.

3.3.1.1 Cipher Suites for TLS 1.2 and Earlier Versions

The first revision of this guidance required support for a small set of cipher suites to promote interoperability and align with TLS specifications. There are no longer any mandatory cipher suite requirements. Cipher suites that comprise AES and other NIST-approved algorithms are acceptable to use, although they are not necessarily equal in terms of security. Cipher suites that use TDEA (3DES) are no longer allowed, due to the limited amounts of data that can be processed under a single key. The server **shall** be configured to only use cipher suites for which it has a valid certificate containing a signature providing at least 112 bits of security.

By removing requirements that specific cipher suites be supported, system administrators have

[19] The full list of IANA values for TLS parameters can be found at https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml.

[20] Perfect forward secrecy is the condition in which the compromise of a long-term private key used in deriving a session key subsequent to the derivation does not cause the compromise of the session key.

more freedom to meet the needs of their environment and applications. It also increases agility by allowing administrators to immediately disable cipher suites when attacks are discovered without breaking compliance.

If a subset of the cipher suites that are acceptable for the server certificate(s) are supported, the following list gives general guidance on choosing the strongest options:

1. Prefer ephemeral keys over static keys (i.e., prefer DHE over DH, and prefer ECDHE over ECDH). Ephemeral keys provide perfect forward secrecy.
2. Prefer GCM or CCM modes over CBC mode. The use of an authenticated encryption mode prevents several attacks (see Section 3.3.2 for more information). Note that these are not available in versions prior to TLS 1.2.
3. Prefer CCM over CCM_8. The latter contains a shorter authentication tag, which provides a lower authentication strength.

This list does not have to be strictly followed, as some environments or applications may have special circumstances. Note that this list may become outdated if an attack emerges on one of the preferred components. If an attack significantly impacts the recommended cipher suites, NIST will address the issue in an announcement on the NIST Computer Security Resource Center website (https://csrc.nist.gov).

3.3.1.1.1 Cipher Suites for ECDSA Certificates

TLS version 1.2 includes authenticated encryption modes, and support for the SHA-256 and SHA-384 hash algorithms, which are not supported in prior versions of TLS. These cipher suites are described in [53] and [49]. TLS 1.2 servers that are configured with ECDSA certificates may be configured to support the following cipher suites, which are only supported by TLS 1.2:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM (0xC0, 0xAC)
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM (0xC0, 0xAD)
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (0xC0, 0xAE)
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8 (0xC0, 0xAF)
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x23)
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x24)

TLS servers may be configured to support the following cipher suites when ECDSA certificates are used with TLS versions 1.2, 1.1, or 1.0:

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA[21] (0xC0, 0x09)

[21] In TLS versions 1.0 and 1.1, DHE and ECDHE cipher suites use SHA-1 for signature generation on the ephemeral parameters (including keys) in the ServerKeyExchange message. While the use of SHA-1 for digital signature generation is generally disallowed by [10], exceptions can be granted by protocol-specific guidance. SHA-1 is allowed for generating digital signatures on ephemeral parameters in TLS. Due to the random nature of the ephemeral keys, a third party is unlikely to cause effective collision. The server and client do not have anything to gain by causing a collision for the connection. Because of the client random and server random values, the server, the client, or a third party cannot use a colliding set of

- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xC0, 0x0A)

3.3.1.1.2 Cipher Suites for RSA Certificates

TLS 1.2 servers that are configured with RSA certificates may be configured to support the following cipher suites:

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x00, 0x9E)
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x00, 0x9F)
- TLS_DHE_RSA_WITH_AES_128_CCM (0xC0, 0x9E)
- TLS_DHE_RSA_WITH_AES_256_CCM (0xC0, 0x9F)
- TLS_DHE_RSA_WITH_AES_128_CCM_8 (0xC0, 0xA2)
- TLS_DHE_RSA_WITH_AES_256_CCM_8 (0xC0, 0xA3)
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x27)
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x00, 0x67)
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x00, 0x6B)

TLS servers may be configured to support the following cipher suites when RSA certificates are used with TLS versions 1.2, 1.1, or 1.0:

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xC0, 0x13)
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xC0, 0x14)
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x00, 0x33)
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x00, 0x39)

3.3.1.1.3 Cipher Suites for DSA Certificates

TLS 1.2 servers that are configured with DSA certificates may be configured to support the following cipher suites:

- TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 (0x00, 0xA2)
- TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 (0x00, 0xA3)
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 (0x00, 0x40)
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 (0x00, 0x6A)

TLS servers may be configured to support the following cipher suites when DSA certificates are used with TLS versions 1.2, 1.1, or 1.0:

- TLS_DHE_DSS_WITH_AES_128_CBC_SHA (0x00, 0x32)

messages to masquerade as the client or server in future connections. Any modification to the parameters by a third party during the handshake will ultimately result in a failed connection.

- TLS_DHE_DSS_WITH_AES_256_CBC_SHA (0x00, 0x38)

3.3.1.1.4 Cipher Suites for DH Certificates

DH certificates contain a static key, and are signed using either DSA or RSA. Unlike cipher suites that use ephemeral DH, these cipher suites contain static DH parameters. While the use of static keys is technically acceptable, the use of ephemeral key cipher suites is encouraged and preferred over the use of the cipher suites listed in this section.

TLS 1.2 servers that are configured with DSA-signed DH certificates may be configured to support the following cipher suites:

- TLS_DH_DSS_WITH_AES_128_GCM_SHA256 (0x00, 0xA4)
- TLS_DH_DSS_WITH_AES_256_GCM_SHA384 (0x00, 0xA5)
- TLS_DH_DSS_WITH_AES_128_CBC_SHA256 (0x00, 0x3E)
- TLS_DH_DSS_WITH_AES_256_CBC_SHA256 (0x00, 0x68)

TLS servers may be configured to support the following cipher suites when DSA-signed DH certificates are used with TLS versions 1.2, 1.1, or 1.0:

- TLS_DH_DSS_WITH_AES_128_CBC_SHA (0x00, 0x30)
- TLS_DH_DSS_WITH_AES_256_CBC_SHA (0x00, 0x36)

TLS 1.2 servers that are configured with RSA-signed DH certificates may be configured to support the following cipher suites:

- TLS_DH_RSA_WITH_AES_128_GCM_SHA256 (0x00, 0xA0)
- TLS_DH_RSA_WITH_AES_256_GCM_SHA384 (0x00, 0xA1)
- TLS_DH_RSA_WITH_AES_128_CBC_SHA256 (0x00, 0x3F)
- TLS_DH_RSA_WITH_AES_256_CBC_SHA256 (0x00, 0x69)

TLS servers may be configured to support the following cipher suites when RSA-signed DH certificates are used with TLS versions 1.2, 1.1, or 1.0:

- TLS_DH_RSA_WITH_AES_128_CBC_SHA (0x00, 0x31)
- TLS_DH_RSA_WITH_AES_256_CBC_SHA (0x00, 0x37)

3.3.1.1.5 Cipher Suites for ECDH Certificates

ECDH certificates contain a static key, and are signed using either ECDSA or RSA. Unlike cipher suites that use ephemeral ECDH, these cipher suites contain static ECDH parameters. The use of ephemeral key cipher suites is encouraged and preferred over the use of the cipher suites listed in this section.

TLS 1.2 servers that are configured with ECDSA-signed ECDH certificates may be configured to support the following cipher suites:

- TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2D)
- TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2E)
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x25)
- TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x26)

TLS servers may be configured to support the following cipher suites when ECDSA-signed ECDH certificates are used with TLS versions 1.2, 1.1, or 1.0:

- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xC0, 0x04)
- TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xC0, 0x05)

TLS 1.2 servers that are configured with RSA-signed ECDH certificates may be configured to support the following cipher suites:

- TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x31)
- TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x32)
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x29)
- TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x2A)

TLS servers may be configured to support the following cipher suites when RSA-signed ECDH certificates are used with TLS versions 1.2, 1.1, or 1.0:

- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xC0, 0x0E)
- TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xC0, 0x0F)

3.3.1.2 Cipher Suites for TLS 1.3

TLS 1.3 servers may be configured to support the following cipher suites:

- TLS_AES_128_GCM_SHA256 (0x13, 0x01)
- TLS_AES_256_GCM_SHA384 (0x13, 0x02)
- TLS_AES_128_CCM_SHA256 (0x13, 0x04)
- TLS_AES_128_CCM_8_SHA256 (0x13, 0x05)

These cipher suites may be used with either RSA or ECDSA server certificates; DSA and DH certificates cannot be used with TLS 1.3. These cipher suites may also be used with pre-shared keys, as specified in Appendix C.

3.3.2 Implementation Considerations

System administrators need to fully understand the ramifications of selecting cipher suites and configuring applications to support only those cipher suites. The security guarantees of the cryptography are limited to the weakest cipher suite supported by the configuration. When configuring an implementation, there are several factors that affect the selection of supported cipher suites.

RFC 4346 [23] describes timing attacks on CBC cipher suites, as well as mitigation techniques. TLS implementations **shall** use the bad_record_mac error to indicate a padding error when communications are secured using a CBC cipher suite. Implementations **shall** compute the MAC regardless of whether padding errors exist.

In addition to the CBC attacks addressed in RFC 4346 [23], the Lucky 13 attack [1] demonstrates that a constant-time decryption routine is also needed to prevent timing attacks. TLS implementations **should** support constant-time decryption, or near constant-time decryption.

The POODLE attack exploits nondeterministic padding in SSL 3.0 [42]. The vulnerability does not exist in the TLS protocols, but the vulnerability can exist in a TLS implementation when the SSL decoder code is reused to process TLS data [37]. TLS implementations **shall** correctly decode the CBC padding bytes.

Note that CBC-based attacks can be prevented by using AEAD cipher suites (e.g., GCM, CCM), which are supported in TLS 1.2.

3.3.2.1 Algorithm Support

Many TLS servers and clients support cipher suites that are not composed of only NIST-approved algorithms. Therefore, it is important that the server is configured to only use NIST-recommended cipher suites. This is particularly important for server implementations that do not allow the server administrator to specify preference order. In such servers, the only way to ensure that a server uses NIST-approved algorithms is to disable cipher suites that use other algorithms.

If the server implementation does allow the server administrator to specify a preference, the system administrator is encouraged to use the preference recommendations listed in Section 3.3.1.1.

3.3.3 Validated Cryptography

The cryptographic module used by the server **shall** be a FIPS 140-validated cryptographic module [66]. All cryptographic algorithms that are included in the configured cipher suites **shall** be within the scope of the validation, as well as the random number generator. Note that the TLS 1.1 pseudorandom function (PRF) uses MD5 and SHA-1 in parallel so that if one hash function is broken, security is not compromised. While MD5 is not a NIST-approved algorithm, the TLS 1.1 PRF is specified as acceptable in SP 800-135 [20]. TLS 1.3 uses the HMAC-based Extract-and-Expand Key Derivation Function (HKDF), described in RFC 5869 [36], to derive the session keys. Note that in TLS versions prior to 1.2, the use of SHA-1 is considered acceptable for signing ephemeral keys and for signing for client authentication. This is due the difficulty for a third party to cause a collision that is not detected. In TLS 1.2, the default hash function in the PRF is SHA-256. Other than the SHA-1 exception listed for specific instances above, all cryptography used **shall** provide at least 112 bits of security. All server and client certificates **shall** contain public keys that offer at least 112 bits of security. All server and client certificates and certificates in their certification paths **shall** be signed using key pairs that offer at least 112 bits of security and SHA-224 or a stronger hashing algorithm. All ephemeral keys used by the client and server **shall** offer at least 112 bits of security. All symmetric algorithms used to protect the TLS data **shall** use keys that offer at least 112 bits of security.

The FIPS 140 validation certificate for the cryptographic module used by the server **shall** indicate that the random bit generator (RBG) has been validated in accordance with the SP 800-90 series [8, 43, 60].[22]

The server random value, sent in the ServerHello message, contains a 4-byte timestamp[23] value and 28-byte random value in TLS version 1.0, 1.1, and 1.2, and contains a 32-byte random value in TLS 1.3. The validated random number generator **shall** be used to generate the random bytes of the server random value.[24] The validated random number generator **should** be used to generate the 4-byte timestamp of the server random value.

3.4 TLS Extension Support

Several TLS extensions are described in RFCs. This section contains recommendations for a subset of the TLS extensions that the Federal agencies **shall**, **should**, or **should not** use as they become prevalent in commercially available TLS servers and clients.

System administrators must carefully consider the risks of supporting extensions that are not listed as mandatory. Only extensions whose specification have an impact on security are discussed here, but the reader is advised that supporting any extension can have unintended security consequences. In particular, enabling extensions increases the potential for implementation flaws and could leave a system vulnerable. For example, the Heartbleed bug [69] was a flaw in an implementation of the heartbeat extension [56]. Although the extension has no inherent security implications, the implementation flaw exposed server data, including private keys, to attackers.

In general, it is advised that servers only be configured to support extensions that are required by the application or enhance security. Extensions that are not needed **should not** be enabled.

3.4.1 Mandatory TLS Extensions

The server **shall** support the use of the following TLS extensions.

5. Renegotiation Indication
6. Server Name Indication
7. Extended Master Secret
8. Signature Algorithms
9. Certificate Status Request extension

[22] Validation will include compliance with SP 800-90C once it is available.

[23] The timestamp value does not need to be correct in TLS. It can be any 4-byte value, unless otherwise restricted by higher-level or application protocols.

[24] TLS 1.3 implementations include a downgrade protection mechanism embedded in the random value that overwrites the last eight bytes of the server random value with a fixed value. When negotiating TLS 1.2, the last eight bytes of the server random will be set to 44 4F 57 4E 47 52 44 01. When TLS 1.1 or below is negotiated, the last eight bytes of the random value will be set to 44 4F 57 4E 47 52 44 00. This overwrite is separate from the validated random bit generator.

3.4.1.1 Renegotiation Indication

Applies to TLS versions: 1.0, 1.1, 1.2

In TLS versions 1.0 to 1.2, session renegotiation is vulnerable to an attack in which the attacker forms a TLS connection with the target server, injects content of its choice, and then splices in a new TLS connection from a legitimate client. The server treats the legitimate client's initial TLS handshake as a renegotiation of the attacker's negotiated session and thus believes that the initial data transmitted by the attacker is from the legitimate client. The session renegotiation extension is defined to prevent such a session splicing or session interception. The extension uses the concept of cryptographically binding the initial session negotiation and session renegotiation.

Server implementations **shall** perform initial and subsequent renegotiations in accordance with RFC 5746 [51] and RFC 8446 [50].

3.4.1.2 Server Name Indication

Applies to TLS versions: 1.0, 1.1, 1.2, 1.3

Multiple virtual servers may exist at the same network address. The server name indication extension allows the client to specify which of the servers located at the address it is trying to connect with. This extension is available in all versions of TLS. The server **shall** be able to process and respond to the server name indication extension received in a ClientHello message as described in [28].

3.4.1.3 Extended Master Secret

Applies to TLS versions: 1.0, 1.1, 1.2

Bhargavan et al. have shown that an active attacker can synchronize two TLS sessions such that they share the same master secret, thus allowing the attacker to perform a man-in-the-middle attack [12]. The Extended Master Secret extension, specified in RFC 7627 [35], prevents such attacks by binding the master secret to a hashed log of the full handshake. The server **shall** support the use of this extension.

3.4.1.4 Signature Algorithms

Applies to TLS versions: 1.2, 1.3

Servers **shall** support the processing of the signature algorithms extension received in a ClientHello message. The extension, its syntax, and processing rules are described in Sections 7.4.1.4.1, 7.4.2, and 7.4.3 of RFC 5246 [24] and Section 4.2.3 of RFC 8446 [50]. Note that the extension described in RFC 8446 updates the extension described in RFC 5246 by adding an additional signature scheme.

3.4.1.5 Certificate Status Request

Applies to TLS versions: 1.0, 1.1, 1.2, 1.3

When the client wishes to receive the revocation status of the TLS server certificate from the TLS server, the client includes the Certificate Status Request (status_request) extension in the ClientHello message. Upon receipt of the status_request extension, a server with a certificate issued by a CA that supports OCSP **shall** include the certificate status along with its certificate by sending a CertificateStatus message immediately following the Certificate message.[25] While the extension itself is extensible, only OCSP-type certificate status is defined in [28]. This extension is also called OCSP stapling.

3.4.2 Conditional TLS Extensions

Support the use of the following TLS extensions under the circumstances described in the following paragraphs:

1. The Fallback Signaling Cipher Suite Value (SCSV) **shall** be supported if the server supports versions of TLS prior to TLS 1.2 and does not support TLS 1.3.
2. The Supported Groups extension **shall be** supported if the server supports ephemeral ECDH cipher suites or if the server supports TLS 1.3.
3. The Key Share extension **shall be** supported if the server supports TLS 1.3.
4. The EC Point Format extension **shall** be supported if the server supports EC cipher suites.
5. The Multiple Certificate Status extension **should** be supported if status information for the server's certificate is available via OCSP, and the extension is supported by the server implementation.
6. The Trusted CA Indication extension **shall** be supported if the server communicates with memory-constrained clients (e.g., low-memory client devices in the Internet of Things), and the server has been issued certificates by multiple CAs.
7. The Encrypt-then-MAC extension **shall** be supported if the server is configured to negotiate CBC cipher suites.
8. The Truncated HMAC extension may be supported if the server communicates with constrained device clients, cipher suites that use CBC mode are supported, and the server implementation does not support variable-length padding.
9. The Pre-Shared Key extension may be supported if the server supports TLS 1.3.
10. The Pre-Shared Key Exchange Modes extension **shall** be supported if the server supports TLS 1.3 and the Pre-Shared Key extension.
11. The Supported Versions extension **shall** be supported if the server supports TLS 1.3.
12. The Cookie extension **shall** be supported if the server supports TLS 1.3.
13. The Certificate Signature Algorithms Extension **shall** be supported if the server supports TLS 1.3, and **should** be supported for TLS 1.2.
14. The Signed Certificate Timestamps extension **should** be supported if the server's certificate was issued by a publicly trusted CA, and the certificate does not include a Signed Certificate Timestamps List extension.

[25] In TLS 1.3 the server includes the certificate status in the Certificate message.

3.4.2.1 Fallback Signaling Cipher Suite Value (SCSV)

Applies to TLS versions: 1.0, 1.1, 1.2

TLS 1.3 includes a downgrade protection mechanism that previous versions do not. In versions prior to TLS 1.3, an attacker can use an external version negotiation as a means to force unnecessary protocol downgrades on a connection. In particular, the attacker can make it appear that the connection failed with the requested TLS version, and some client implementations will try the connection again with a downgraded protocol version. This cipher suite value, described in RFC 7507 [41], provides a mechanism to prevent unintended protocol downgrades in versions prior to TLS 1.3. Clients signal when a connection is a fallback, and if the server deems it inappropriate (i.e., the server supports a higher TLS version), the server returns a fatal alert.

When TLS versions prior to TLS 1.2 are supported by the server, and TLS version 1.3 is not supported, the fallback SCSV **shall** be supported.

3.4.2.2 Supported Groups

Applies to TLS versions: 1.0, 1.1, 1.2, 1.3

The Supported Groups extension (supported_groups) allows the client to indicate the domain parameter groups that it supports to the server. The extension was originally called the Supported Elliptic Curves extension (elliptic_curves), and was only used for elliptic curve groups, but it may now also be used to negotiate finite field groups. In TLS 1.3, the Supported Groups extension must be used to negotiate both elliptic curve and finite field groups. Servers that support either ephemeral ECDH cipher suites or TLS 1.3 **shall** support this extension. When elliptic curve cipher suites are configured, at least one of the NIST-approved curves, P-256 (secp256r1) and P-384 (secp384r1), **shall** be supported as described in RFC 8422 [44]. Additional NIST-recommended elliptic curves are listed in SP 800-56A, Appendix D [7]. Finite field groups that are approved for TLS in SP 800-56A, Appendix D may be supported.

3.4.2.3 Key Share

Applies to TLS version 1.3

The Key Share extension is used in TLS 1.3 to send cryptographic parameters. Servers that support TLS 1.3 **shall** support this extension as described in Section 4.2.7 of RFC 8446 [50].

3.4.2.4 Supported Point Formats

Applies to TLS versions: 1.0, 1.1, 1.2

Servers that support EC cipher suites with TLS 1.2 and below **shall** be able to process the supported point format received in the ClientHello message by the client. The servers **shall** process this extension in accordance with Section 5.1 of RFC 8422 [44].

Servers that support EC cipher suites **shall** also be able to send the supported EC point format in the ServerHello message as described in Section 5.2 of RFC 8422 [44].

3.4.2.5 Multiple Certificate Status

Applies to TLS versions: 1.0, 1.1, 1.2

The multiple certificate status extension improves on the Certificate Status Request extension described in Section 3.4.1.5 by allowing the client to request the status of all certificates provided by the server in the TLS handshake. When the server returns the revocation status of all the certificates in the server certificate chain, the client does not need to query any revocation service providers, such as OCSP responders. This extension is documented in RFC 6961 [47]. Servers that have this capability and that have certificates issued by CAs that support OCSP **should** be configured to support this extension.

3.4.2.6 Trusted CA Indication

Applies to TLS versions: 1.0, 1.1, 1.2

The trusted CA indication (trusted_ca_keys) extension allows a client to specify which CA root keys it possesses. This is useful for sessions where the client is memory-constrained and possesses a small number of root CA keys. Servers that communicate with memory-constrained clients and that have been issued certificates by multiple CAs **shall** be able to process and respond to the trusted CA indication extension received in a ClientHello message as described in [28].

3.4.2.7 Encrypt-then-MAC

Applies to TLS versions: 1.0, 1.1, 1.2

Several attacks on CBC cipher suites have been possible due to the MAC-then-encrypt order of operations used in TLS versions 1.0, 1.1, and 1.2. The Encrypt-then-MAC extension alters the order that the encryption and MAC operations are applied to the data. This is believed to provide stronger security, and mitigate or prevent several known attacks on CBC cipher suites. Servers that are configured to negotiate CBC cipher suites **shall** support this extension as described in [32].

3.4.2.8 Truncated HMAC

Applies to TLS versions: 1.0, 1.1, 1.2

The Truncated HMAC extension allows a truncation of the HMAC output to 80 bits for use as a MAC tag. An 80-bit MAC tag complies with the recommendations in SP 800-107 [19], but reduces the security provided by the integrity algorithm. Because forging a MAC tag is an online attack, and the TLS session will terminate immediately when an invalid MAC tag is encountered, the risk introduced by using this extension is low. However, truncated MAC tags **shall not** be used in conjunction with variable-length padding, due to attacks described by Paterson et al. [46]. This extension is only applicable when cipher suites that use CBC modes are supported.

3.4.2.9 Pre-Shared Key

Applies to TLS version 1.3

The Pre-Shared Key extension (pre_shared_key), available in TLS 1.3, is used to indicate the identity of the pre-shared key to be used for PSK key establishment. In TLS 1.3 pre-shared keys may either be established out-of-band, as in TLS 1.2 are below, or in a previous connection, in which case they are used for session resumption. Servers that support TLS 1.3 may be configured to support this extension in order to support session resumption or to support the use of pre-shared keys that are established out-of-band.

3.4.2.10 Pre-Shared Key Exchange Modes

Applies to TLS version 1.3

A TLS 1.3 client must send the Pre-Shared Key Exchange Modes extension (psk_key_exchange_modes) if it sends the Pre-Shared Key extension. TLS 1.3 servers use the list of key exchange modes present in the extension to select an appropriate key exchange method. TLS servers that support TLS 1.3 and the Pre-Shared Key extension **shall** support this extension.

3.4.2.11 Supported Versions

Applies to TLS version 1.3

The supported versions extension is sent in the ClientHello message to indicate which versions of TLS the client supports. A TLS 1.3 server **shall** be able to process this extension. When it is absent from the ClientHello message, the server **shall** use the version negotiation specified in TLS 1.2 and earlier.

3.4.2.12 Cookie

Applies to TLS version 1.3

The cookie extension allows the server to force the client to prove that it is reachable at its apparent network address, and offload state information to the client. Servers that support TLS 1.3 may support the cookie extension in accordance with RFC 8446 [50].

3.4.2.13 Certificate Signature Algorithms

Applies to TLS versions: 1.2, 1.3

The Certificate Signature Algorithms extension (signature_algorithms_cert) indicates the signature algorithms that may be used in certificates. (When it is not present, algorithms in the Signature Algorithms extension apply to certificates as well.) TLS servers that support TLS 1.3 **shall** support this extension, and it **should** be supported for TLS 1.2.

3.4.2.14 Signed Certificate Timestamps

Applies to TLS versions: 1.0, 1.1, 1.2, 1.3

The Certificate Transparency project (described in RFC 6962 [39]) strives to reduce the impact of certificate-based threats by making the issuance of CA-signed certificates more transparent. This is done through the use of public logs of certificates, public log monitoring, and public certificate auditing. Certificate logs are cryptographically assured records of certificates that are open to public scrutiny. Certificates may be appended to logs, but they cannot be removed, modified, or inserted into the middle of a log. Monitors watch certificate logs for suspicious certificates, such as those that were not authorized by the domain they claim to represent. Auditors have the ability to check the membership of a particular certificate in a log, as well as verify the integrity and consistency of logs.

Evidence that the server's certificate has been submitted to Certificate Transparency logs may be provided to clients either in the certificate itself or in a Signed Certificate Timestamps TLS extension (signed_certificate_timestamp). Servers with certificates issued by publicly trusted CAs that do not include a Signed Certificate Timestamps List extension **should** support the Signed Certificate Timestamps TLS extension.

3.4.3 Discouraged TLS Extensions

The following extensions **should not** be used:

1. Client Certificate URL
2. Early Data Indication

The Raw Public Keys extension **shall not** be supported.

3.4.3.1 Client Certificate URL

Applies to TLS versions: 1.0, 1.1, 1.2

The Client Certificate URL extension allows a client to send a URL pointing to a certificate, rather than sending a certificate to the server during mutual authentication. This can be very useful for mutual authentication with constrained clients. However, this extension can be used for malicious purposes. The URL could belong to an innocent server on which the client would like to perform a denial of service attack, turning the TLS server into an attacker. A server that supports this extension also acts as a client while retrieving a certificate, and therefore becomes subject to additional security concerns. For these reasons, the Client Certificate URL extension **should not** be supported. However, if an agency determines that the risks are minimal, and this extension is needed for environments where clients are in constrained devices, the extension may be supported. If the client certificate URL extension is supported, the server **shall** be configured to mitigate the security concerns described above and in Section 11.3 of [28].

3.4.3.2 Early Data Indication

Applies to TLS version 1.3

The Early Data Indication extension (early_data) allows the client to send application data in the ClientHello message when pre-shared keys are used. This includes pre-shared keys that are established out-of-band, as well as those used for session resumption. TLS does not protect this early data against replay attacks. Servers **should not** process early data received in the ClientHello message. If the server is configured to send the Early Data Indication extension, the server **shall** use methods of replay protection, such as those described in Section 8 of RFC 8446 [50]. See Section 3.6 for more information on early data (also called 0-RTT data).

3.4.3.3 Raw Public Keys

Applies to TLS versions: 1.0, 1.1 1.2, 1.3

The Raw Public Keys extension, described in RFC 7250 [45], provides an alternative to certificate-based authentication that only uses the information contained in the SubjectPublicKeyInfo field an X.509 version 3 certificate. While this reduces the size of the public key structure and simplifies processing, it removes any assurances that a public key belongs to a particular entity. To provide authentication, an out-of-band binding between public key and entity must be used.

3.5 Client Authentication

Where strong cryptographic client authentication is required, TLS servers may use the TLS protocol client authentication option to request a certificate from the client to cryptographically authenticate the client.[26] For example, the Personal Identity Verification (PIV) Authentication certificate [64] (and the associated private key) provides a suitable option for strong authentication of Federal employees and contractors. To ensure that agencies are positioned to take full advantage of the PIV Card, all TLS servers that perform client authentication **shall** implement certificate-based client authentication.

The client authentication option requires the server to implement the X.509 path validation mechanism and a trust anchor store. Requirements for these mechanisms are specified in Sections 3.5.1 and 3.5.2, respectively. To ensure that cryptographic authentication actually results in strong authentication, client keys **shall** contain at least 112 bits of security. Section 3.5.3 describes mechanisms that can contribute, albeit indirectly, to enforcing this requirement. Section 3.5.4 describes the client's use of the server hints list.

The TLS server **shall** be configurable to terminate the connection with a fatal "handshake failure" alert when a client certificate is requested, and the client does not have a suitable certificate.

[26] The CertificateVerify message is sent to explicitly verify a client certificate that has a signing capability. In TLS 1.1 (and TLS 1.0), this message uses SHA-1 to generate a signature on all handshake messages that came before it. SP 800-131A [10] states that the use of SHA-1 for digital signature generation is disallowed after 2013. Even if a collision is found, the client must use its private key to authenticate itself by signing the hash. Due to the client random and server random values, the server, the client, or a third party cannot use a colliding set of messages to masquerade as the client or server in future connections. Any modification to this message, preceding messages, or subsequent messages will ultimately result in a failed connection. Therefore, SHA-1 is allowed for generating digital signatures in the TLS CertificateVerify message.

3.5.1 Path Validation

The client certificate **shall** be validated in accordance with the certification path validation rules specified in Section 6 of [18]. In addition, the revocation status of each certificate in the certification path **shall** be validated using the Online Certificate Status Protocol (OCSP) or a certificate revocation list (CRL). OCSP checking **shall** be in compliance with RFC 6960 [55].

Revocation information **shall** be obtained as described in Section 3.2.2.

The server **shall** be able to determine the certificate policies that the client certificate is trusted for by using the certification path validation rules specified in Section 6 of [18]. Server and backend applications may use this determination to accept or reject the certificate. Checking certificate policies assures the server that only client certificates that have been issued with acceptable assurance, in terms of CA and registration system and process security, are accepted.

Not all commercial products may support the public-key certification path validation and certificate policy processing rules listed and cited above. When implementing client authentication, the Federal agencies **shall** either use the commercial products that meet these requirements or augment commercial products to meet these requirements.

The server **shall** be able to provide the client certificate, and the certificate policies for which the client certification path is valid, to the applications in order to support access control decisions.

3.5.2 Trust Anchor Store

Having an excessive number of trust anchors installed in the TLS application can expose the application to all the PKIs emanating from those trust anchors. The best way to minimize the exposure is to only include the trust anchors in the trust anchor store that are absolutely necessary for client public-key certificate authentication.

The server **shall** be configured with only the trust anchors that the server trusts, and of those, only the ones that are required to authenticate the clients, in the case where the server supports client authentication in TLS. These trust anchors are typically a small subset of the trust anchors that may be included on the server by default. Also, note that this trust anchor store is distinct from the machine trust anchor store. Thus, the default set of trust anchors **shall** be examined to determine if any of them are required for client authentication. Some specific enterprise and/or PKI service provider trust anchor may need to be added.

In the U.S. federal environment, in most situations, the Federal Common Policy Root or the agency root (if cross certified with the Federal Bridge Certification Authority or the Federal Common Policy Root) should be sufficient to build a certification path to the client certificates.

System administrators of a TLS server that supports certificate-based client authentication **shall** perform an analysis of the client certificate issuers and use that information to determine the minimum set of trust anchors required for the server. The server **shall** be configured to only use those trust anchors.

3.5.3 Checking the Client Key Size

The only direct mechanism for a server to check whether the key size and algorithms presented in a client public-key certificate are acceptable is for the server to examine the public key and algorithm in the client's certificate. An indirect mechanism is to check that the certificate policies extension in the client public-key certificate indicates the minimum cryptographic strength of the signature and hashing algorithms used, and for the server to perform certificate policy processing and checking. The server **shall** check the client key length if client authentication is performed, and the server implementation provides a mechanism to do so. Federal Agencies **shall** use the key size guidelines provided in SP 800-131A [10] to check the client key size.

3.5.4 Server Hints List

Clients may use the list of trust anchors sent by the server in the CertificateRequest message to determine if the client's certification path terminates at one of these trust anchors. The list sent by the server is known as a "hints list." When the server and client are in different PKI domains, and the trust is established via direct cross-certification between the two PKI domains (i.e., the server PKI domain and the client PKI domain) or via transitive cross-certification (i.e., through cross-certifications among multiple PKI domains), the client may erroneously decide that its certificate will not be accepted by the server since the client's trust anchor is not sent in the hints list. To mitigate this failure, the server **shall** either 1) maintain the trust anchors of the various PKIs whose subscribers are the potential clients for the server, and include them in the hints list, or 2) be configured to send an empty hints list so that the client can always provide a certificate it possesses. The hints list **shall** be distinct from the server's trust anchor store.[27] In other words, the server **shall** continue to only populate its trust anchor store with the trust anchor of the server's PKI domain and the domains it needs to trust directly for client authentication. Note that the distinction between the server hints list and the server's own trust store is as follows: 1) the hints list is the list of trust anchors that a potential client might trust; and 2) the server's trust store is the list of trust anchors that the server explicitly trusts.

3.6 Session Resumption and Early Data

Previous TLS sessions can be resumed, allowing for a connection to be established using an abbreviated handshake. All versions of TLS offer session resumption, although the mechanism for performing resumption differs. A server may be configured to ignore requests to resume a session, if the implementation allows it.

Additional mechanisms have been developed for session resumption, such as the Stateless TLS Session Resumption extension [54]. While these guidelines neither encourage or discourage the use of such mechanisms, it is important to understand the security impact if long term or shared keys are compromised. If resumption is allowed, frequent key rotation and short lifetimes for resumption information are recommended, as applicable. See [58] for discussion on the security

[27] Depending on the server and client trust anchors, the two lists could be identical, could have some trust anchors in common, or have no trust anchors in common.

impacts of resumption mechanisms.

TLS 1.3 allows the client to send data in the first flight of handshake, known as 0-RTT data. This practice may provide opportunities for attackers, such as replay attacks.[28] The TLS 1.3 specification describes two mechanisms to mitigate threats introduced by 0-RTT data. One of these mechanisms is single-use tickets, which allows each session ticket to be used only once. It may be difficult to implement this mechanism in an environment with distributed servers, as a session database must be shared between servers. ClientHello recording is a second mechanism that defends against replay attacks by recording a unique value derived from the ClientHello and rejecting duplicates. To limit the size of the list, the server can maintain a list only within a specified time window. In general, 0-RTT data **should not** be accepted by the server. If the server does allow 0-RTT data, then the server **should** use the single-use ticket mechanism in accordance with RFC 8446 (see Section 8 of [50]).

3.7 Compression Methods

The use of compression may enable attackers to perform attacks using compression-based side channels (e.g., [52], [11]). To defend against these attacks, the null compression method **shall** be enabled, and all other compression methods **shall** be disabled.

3.8 Operational Considerations

The sections above specify TLS-specific functionality. This functionality is necessary, but is not sufficient, to achieve security in an operational environment.

Federal agencies **shall** ensure that TLS servers include appropriate network security protections as specified in other NIST guidelines, such as SP 800-53 [34].

The server **shall** operate on a secure operating system.[29] Where the server relies on a FIPS 140 Level 1 cryptographic module, the software and private key **shall** be protected using the operating system identification, authentication and access control mechanisms. In some highly sensitive applications, server private keys may require protection using a FIPS 140 Level 2 or higher hardware cryptographic module.

The server and associated platform **shall** be kept up-to-date in terms of security patches. This is critical to various aspects of security.

---

[28] TLS does not inherently provide replay protection for 0-RTT data.

[29] A secure operating system contains and uses the following features: operating system protection from applications and processes; operating system mediated isolation among applications and processes; user identification and authentication; access control based on authenticated user identity, and event logging of security-relevant activities.

4 Minimum Requirements for TLS Clients

This section provides a minimum set of requirements that a TLS client must meet in order to adhere to these guidelines. Requirements are organized as follows: TLS protocol version support; client keys and certificates; cryptographic support; TLS extension support; server authentication; session resumption; compression methods; and operational considerations.

Specific requirements are stated as either implementation requirements or configuration requirements. Implementation requirements indicate that Federal agencies **shall not** procure TLS client implementations unless they include the required functionality. Configuration requirements indicate that system administrators are required to verify that particular features are enabled, or in some cases, configured appropriately if present.

4.1 Protocol Version Support

The client **shall** be configured to use TLS 1.2 and **should** be configured to use TLS 1.3. The client may be configured to use TLS 1.1 and TLS 1.0 to facilitate communication with private sector servers. The client **shall not** be configured to use SSL 2.0 or SSL 3.0. Agencies **shall** support TLS 1.3 by January 1, 2024. After this date clients **shall** be configured to use TLS 1.3. Note that TLS 1.3 and 1.2 are intended to coexist, and should both be enabled after the TLS 1.3 adoption deadline.

4.2 Client Keys and Certificates

Some applications may require client authentication. For TLS, this can be achieved by performing mutual authentication using certificates.

4.2.1 Client Certificate Profile

When certificate-based client authentication is needed, the client **shall** be configured with a certificate that adheres to the recommendations presented in this section. A client certificate may be configured on the system or located on an external device (e.g., a PIV Card). For this specification, the TLS client certificate **shall** be an X.509 version 3 certificate; both the public key contained in the certificate and the signature **shall** provide at least 112 bits of security. If the client supports TLS versions prior to TLS 1.2, the certificate **should** be signed with an algorithm that is consistent with the public key:[30]

- Certificates containing RSA (signature), ECDSA, or DSA public keys **should** be signed with those same signature algorithms, respectively;
- Certificates containing Diffie-Hellman certificates **should** be signed with DSA; and
- Certificates containing ECDH public keys **should** be signed with ECDSA.

The client certificate profile is listed in Table 4-1. In the absence of an agency-specific client certificate profile, this profile **should** be used for client certificates.

[30] This recommendation is an artifact of requirements in TLS 1.0 and 1.1.

Table 4-1: TLS Client Certificate Profile

Field	Critical	Value	Description
Version	N/A	2	Version 3
Serial Number	N/A	Unique positive integer	Must be unique
Issuer Signature Algorithm	N/A	*Values by CA key type:*	
		sha256WithRSAEncryption {1 2 840 113549 1 1 11}, or stronger	CA with RSA key
		id-RSASSA-PSS { 1 2 840 113549 1 1 10 }	CA with RSA key
		ecdsa-with-SHA256 {1 2 840 10045 4 3 2}, or stronger	CA with elliptic curve key
		id-dsa-with-sha256 {2 16 840 1 101 3 4 3 2}, or stronger	CA with DSA key
Issuer Distinguished Name	N/A	Unique X.500 Issuing CA DN	A single value **shall** be encoded in each RDN. All attributes that are of directoryString type **shall** be encoded as a printable string.
Validity Period	N/A	3 years or less	Dates through 2049 expressed in UTCTime
Subject Distinguished Name	N/A	Unique X.500 subject DN per agency requirements	A single value **shall** be encoded in each RDN. All attributes that are of directoryString type **shall** be encoded as a printable string.
Subject Public Key Information	N/A	*Values by certificate type:*	
		rsaEncryption {1 2 840 113549 1 1 1}	RSA signature certificate 2048-bit RSA key modulus, or other approved lengths as defined in [FIPS186-4] and [5] Parameters: NULL
		ecPublicKey {1 2 840 10045 2 1}	ECDSA signature certificate or ECDH certificate Parameters: namedCurve OID for names curve specified in SP 800-186.[31] The curve **shall** be P-256 or P-384 SubjectPublic Key: Uncompressed EC Point.
		id-dsa {1 2 840 10040 4 1}	DSA signature certificate Parameters: p, q, g
		dhpublicnumber {1 2 840 10046 2 1}	DH certificate Parameters: p, g, q
Issuer's Signature	N/A	Same value as in Issuer Signature Algorithm	
Extensions			
Authority Key Identifier	No	Octet String	Same as subject key identifier in issuing CA certificate

[31] The recommended elliptic curves now listed in FIPS 186-4 [62] will be moved to SP 800-186. Until SP 800-186 is published, the recommended elliptic curves should be taken from FIPS 186-4.

Field	Critical	Value	Description
			Prohibited: Issuer DN, Serial Number tuple
Subject Key Identifier	No	Octet String	Same as in PKCS-10 request or calculated by the issuing CA
Key Usage	Yes	digitalSignature	RSA certificate, DSA certificate, ECDSA certificate
		keyAgreement	ECDH certificate, DH certificate
Extended Key Usage	No	id-kp-clientAuth {1 3 6 1 5 5 7 3 2}	Required
		anyExtendedKeyUsage {2 5 29 37 0}	The anyExtendedKeyUsage OID **should** be present if the extended key usage extension is included, but there is no intention to limit the types of applications with which the certificate may be used (e.g., the certificate is a general-purpose authentication certificate).
			Prohibited: all others unless consistent with key usage extension
Certificate Policies	No	Per issuer's X.509 certificate policy	
Subject Alternative Name	No	RFC 822 e-mail address, Universal Principal Name (UPN), DNS Name, and/or others	Optional
Authority Information Access	No	id-ad-caIssuers	Required. Access method entry contains HTTP URL for certificates issued to issuing CA
		id-ad-ocsp	Optional. Access method entry contains HTTP URL for the issuing CA OCSP responder
CRL Distribution Points	No	See comments	Optional: HTTP value in distributionPoint field pointing to a full and complete CRL. Prohibited: reasons and cRLIssuer fields, and nameRelativetoCRLIssuer CHOICE

If a client has multiple certificates that meet the requirements of the TLS server, the TLS client (e.g., a browser) may ask the user to select from a list of certificates. The extended key usage (EKU) extension limits the operations for which the keys in a certificate may be used, and so the use of the EKU extension in client certificates may eliminate this request. If the EKU extension is included in client certificates, then the id-kp-client-auth key purpose OID **should** be included in the certificates to be used for TLS client authentication and **should** be omitted from any other certificates.

Client certificates are also filtered by TLS clients on the basis of an ability to build a path to one of the trust anchors in the hints list sent by the server, as described in Section 3.5.4.

4.2.2 Obtaining Revocation Status Information for the Server Certificate

The client **shall** perform revocation checking of the server certificate. Revocation information can be obtained by the client from one of the following locations:

1. OCSP response or responses in the server's CertificateStatus message ([28], [47]) (or Certificate message in TLS 1.3);

2. Certificate Revocation List (CRL) or OCSP response in the client's local certificate store;
3. OCSP response from a locally configured OCSP responder;
4. OCSP response from the OCSP responder location identified in the OCSP field in the Authority Information Access extension in the server certificate; or
5. CRL from the CRL Distribution Point extension in the server certificate.

When the server does not provide the revocation status, the local certificate store does not have the current or a cogent CRL or OCSP response, and the OCSP responder and the CRL distribution point are unavailable or inaccessible at the time of TLS session establishment, the client will either terminate the connection or accept a potentially revoked or compromised certificate. The decision to accept or reject a certificate in this situation **should** be made according to agency policy.

Other emerging concepts that can be useful in lieu of revocation checking are further discussed in Appendix E.2.

4.2.3 Client Public-Key Certificate Assurance

The client public-key certificate may be trusted by the servers on the basis of the policies, procedures and security controls used to issue the client public-key certificate as described in Section 3.5.1. For example, these guidelines recommend that the PIV Authentication certificate be the norm for authentication of Federal employees and long-term contractors. PIV Authentication certificate policy is defined in the Federal PKI Common Policy Framework [30], and PIV-I Authentication certificate policy is defined in the X.509 Certificate Policy for the Federal Bridge Certification Authority [59]. Depending on the requirements of the server-side application, other certificate policies may also be acceptable. Guidance regarding other certificate policies is outside the scope of these guidelines.

4.3 Cryptographic Support

4.3.1 Cipher Suites

The acceptable cipher suites for a TLS client are the same as those for a TLS server. General-purpose cipher suites are listed in Section 3.3.1. Cipher suites appropriate for pre-shared key environments for TLS 1.2 and prior versions are listed in Appendix C. Applications that require RSA key transport as the key exchange may use cipher suites listed in Appendix D during the deprecation period. When ephemeral keys are used to establish the master secret, each ephemeral key-pair (i.e., the server ephemeral key-pair and the client ephemeral key-pair) **shall** have at least 112 bits of security.

The client **should not** be configured to use cipher suites other than those listed in Section 3.3.1, Appendix C, or Appendix D.

To mitigate attacks against CBC mode, TLS implementations that support versions prior to TLS 1.3 **shall** use the bad_record_mac error to indicate a padding error. Implementations **shall** compute the MAC regardless of whether padding errors exist. TLS implementations **should** support constant-time decryption, or near constant-time decryption. This does not apply to TLS

1.3 implementations, as they do not support cipher suites that use CBC mode.

4.3.2 Validated Cryptography

The client **shall** use validated cryptography, as described for the server in Section 3.3.3.

The validated random number generator **shall** be used to generate the random bytes (32 bytes in TLS 1.3; 28 bytes in prior TLS versions) of the client random value. The validated random number generator **should** be used to generate the 4-byte timestamp of the client random value for TLS versions prior to TLS 1.3.

4.4 TLS Extension Support

In general, it is advised that clients only be configured to support extensions that are required for interoperability or enhance security. Extensions that are not needed **should not** be enabled.

4.4.1 Mandatory TLS Extensions

The client **shall** be configured to use the following extensions:

1. Renegotiation Indication
2. Server Name Indication
3. Extended Master Secret
4. Signature Algorithms
5. Certificate Status Request

4.4.1.1 Renegotiation Indication

Applies to TLS versions: 1.0, 1.1, 1.2

The Renegotiation Indication extension is required by these guidelines as described in Section 3.4.1.1. Clients **shall** perform the initial and subsequent renegotiations in accordance with RFC 5746 [51].

4.4.1.2 Server Name Indication

Applies to TLS versions: 1.0, 1.1, 1.2, 1.3

The server name indication extension is described in Section 3.4.1.2. The client **shall** be capable of including this extension in a ClientHello message, as described in RFC 6066 [28].

4.4.1.3 Extended Master Secret

Applies to TLS versions: 1.0, 1.1, 1.2

The Extended Master Secret extension, described in Section 3.4.1.3, prevents man-in-the-middle attacks by binding the master secret to a hashed log of the full handshake. The client **shall** support this extension.

4.4.1.4 Signature Algorithms

Applies to TLS versions: 1.2, 1.3

The clients **shall** assert acceptable hashing and signature algorithm pairs in this extension in TLS 1.2 and TLS 1.3 ClientHello messages. The extension, its syntax, and processing rules are described in Sections 7.4.1.4.1, 7.4.4, 7.4.6 and 7.4.8 of RFC 5246 [24] and in Section 4.2.3 of RFC 8446 [50]. Note that the extension described in RFC 8446 updates the extension described in RFC 5246 by adding an additional signature scheme.

4.4.1.5 Certificate Status Request

Applies to TLS versions: 1.0, 1.1, 1.2, 1.3

The client **shall** include the “status_request” extension in the ClientHello message.

4.4.2 Conditional TLS Extensions

A TLS client supports the following TLS extensions under the circumstances described:

1. The Fallback Signaling Cipher Suite Value (SCSV) **shall** be supported if the client supports versions of TLS prior to TLS 1.2 and does not support TLS 1.3.
2. The Supported Groups extension **shall** be supported if the client supports ephemeral ECDH cipher suites or if the client supports TLS 1.3.
3. The Key Share extension **shall be** supported if the client supports TLS 1.3.
4. The EC Point Format TLS extension **shall** be supported if the client supports EC cipher suite(s).
5. The Multiple Certificate Status extension **should** be enabled if the extension is supported by the client implementation.
6. The Trusted CA Indication extension **should** be supported by clients that run on memory-constrained devices where only a small number of CA root keys are stored.
7. The Encrypt-then-MAC extension **shall** be supported when CBC mode cipher suites are configured.
8. The Truncated HMAC extension may be supported by clients that run on constrained devices when variable-length padding is not supported and cipher suites that use CBC mode are supported.
9. The Pre-Shared Key extension may be supported by TLS 1.3 clients.
10. The Pre-Shared Key Exchange Modes extension **shall** be supported by TLS 1.3 clients that support the Pre-Shared Key extension.
11. The Supported Versions extension **shall** be supported by TLS 1.3 clients.
12. The Cookie extension **shall** be supported by TLS 1.3 clients.
13. The Certificate Signature Algorithms Extension **shall** be supported if the client supports TLS 1.3, and **should** be supported for TLS 1.2.

4.4.2.1 Fallback Signaling Cipher Suite Value (SCSV)

Applies to TLS versions: 1.0, 1.1, 1.2

This cipher suite value, described in Section 3.4.2.1, provides a mechanism to prevent unintended protocol downgrades in TLS versions prior to TLS 1.3. Clients signal when a connection is a fallback, and if the server supports a higher TLS version, the server returns a fatal alert. If the client does not support TLS 1.3, and is attempting to connect with a TLS version prior to TLS 1.2, the client **shall** include TLS_FALLBACK_SCSV at the end of the cipher suite list in the ClientHello message.

4.4.2.2 Supported Groups

Applies to TLS versions: 1.0, 1.1, 1.2, 1.3

The Supported Groups extension (supported_groups) is described in Section 3.4.2.2. Client implementations **shall** send this extension in TLS 1.3 ClientHello messages and in ClientHello messages that include ephemeral ECDH cipher suites. When elliptic curve cipher suites are configured, at least one of the NIST-approved curves, P-256 (secp256r1) and P-384 (secp384r1), **shall** be supported as described in RFC 8422 [44]. Additional NIST-recommended elliptic curves are listed in SP 800-56A, Appendix D [7]. Finite field groups that are approved for TLS in SP 800-56A, Appendix D may be supported.

4.4.2.3 Key Share

Applies to TLS version 1.3

The Key Share extension is used to send cryptographic parameters. Clients that support TLS 1.3 **shall** support this extension as described in Section 4.2.7 of RFC 8446 [50].

4.4.2.4 Supported Point Formats

Applies to TLS versions: 1.0, 1.1, 1.2

The clients that support EC cipher suites with TLS 1.2 and below **shall** be capable of specifying supported point formats in the ClientHello message, in accordance with Section 5.1 of [44].

Clients that support EC cipher suites **shall** support the processing of at least one[32] of the EC point formats received in the ServerHello message, as described in Section 5.2 of [44].

4.4.2.5 Multiple Certificate Status

Applies to TLS versions: 1.0, 1.1, 1.2

The multiple certificate status extension is described in Section 3.4.2.5. This extension improves on the Certificate Status Request extension described in Section 3.4.1.5 by allowing the client to request the status of all certificates provided by the server in the TLS handshake. This extension is documented in RFC 6961 [47]. Client implementations that have this capability **should** be

[32] The uncompressed point format must be supported, and all others are deprecated in TLS, as described in Sections 5.1.2 of RFC 8422 [44].

configured to include this extension in the ClientHello message.

4.4.2.6 Trusted CA Indication

Applies to TLS versions: 1.0, 1.1, 1.2

Clients that run on memory-constrained devices where only a small number of CA root keys are stored **should** be capable of including the trusted CA indication (trusted_ca_keys) extension in a ClientHello message as described in [28].

4.4.2.7 Encrypt-then-MAC

Applies to TLS versions: 1.0, 1.1, 1.2

The Encrypt-then-MAC extension, described in Section 3.4.2.7, can mitigate or prevent several known attacks on CBC cipher suites. In order for this modified order of operations to be applied, both server and client need to implement the Encrypt-then-MAC extension and negotiate its use. When CBC mode cipher suites are configured, clients **shall** support this extension as described in RFC 7366 [32]. The client **shall** include this extension in the ClientHello message whenever the ClientHello message includes CBC cipher suites.

4.4.2.8 Truncated HMAC

Applies to TLS versions: 1.0, 1.1, 1.2

The Truncated HMAC extension is described in Section 3.4.2.8. Clients running on constrained devices may support this extension. The Truncated HMAC extension **shall not** be used in conjunction with variable-length padding, due to attacks described by Paterson et al. [46]. This extension is only applicable when cipher suites that use CBC modes are supported.

4.4.2.9 Pre-Shared Key

Applies to TLS version 1.3

The Pre-Shared Key extension (pre_shared_key) is used to indicate the identity of the pre-shared key to be used for PSK key establishment. In TLS 1.3 pre-shared keys may either be established out-of-band, as in TLS 1.2 and prior versions, or in a previous connection, in which case they are used for session resumption. Clients that support TLS 1.3 may be configured to use this extension in order to allow session resumption or to allow the use of pre-shared keys that are established out-of-band.

4.4.2.10 Pre-Shared Key Exchange Modes

Applies to TLS version 1.3

A TLS 1.3 client must send the Pre-Shared Key Exchange Modes extension (psk_key_exchange_modes) if it sends the Pre-Shared Key extension, otherwise the server will abort the handshake. TLS clients that support TLS 1.3 and the Pre-Shared Key extension **shall** implement this extension.

4.4.2.11 Supported Versions

Applies to TLS version 1.3

The supported versions extension indicates which versions of TLS the client is able to negotiate. A TLS 1.3 client **shall** send this extension in the ClientHello message.

4.4.2.12 Cookie

Applies to TLS version 1.3

The cookie extension allows the server to force the client to prove that it is reachable at its apparent network address, and offload state to the client. Clients that support TLS 1.3 **shall** support the cookie extension in accordance with RFC 8446 [50].

4.4.2.13 Certificate Signature Algorithms

Applies to TLS versions: 1.2, 1.3

The Certificate Signature Algorithms extension (signature_algorithms_cert) indicates the signature algorithms that may be used in certificates. This allows the entity requesting a certificate (client or server) to request different signature algorithms for the certificate than for the TLS handshake. A client may send this extension to the server, and may receive this extension from a server that is requesting certificate-based client authentication. This extension does not need to be sent if the algorithms in the Signature Algorithms extension apply to certificates as well. TLS client implementations that support TLS 1.3 **shall** support this extension, and it **should** be supported for TLS 1.2.

4.4.3 Discouraged TLS Extension

The following extensions **should not** be used:

1. Client Certificate URL
2. Early Data Indication

The Raw Public Key extension **shall not** be supported.

The reasons for discouraging the use of these extensions can be found in Section 3.4.3.

4.5 Server Authentication

The client **shall** be able to build the certification path for the server certificate presented in the TLS handshake with at least one of the trust anchors in the client trust store, if an appropriate trust anchor is present in the store. The client may use all or a subset of the following resources to build the certification path: the local certificate store, certificates received from the server during the handshake, LDAP, the resources declared in CA Repository field of the Subject Information Access extension in various CA certificates, and the resources declared in the CA Issuers field of the Authority Information Access extension in various certificates.

4.5.1 Path Validation

The client **shall** validate the server certificate in accordance with the certification path validation rules specified in Section 6 of [18]. The revocation status of each certificate in the certification path **shall** be checked using the Online Certificate Status Protocol (OCSP) or a certificate revocation list (CRL). OCSP checking **shall** be in compliance with [55]. Revocation information **shall** be obtained as described in Section 4.2.2.

Not all clients support name constraint checking. Federal agencies **should** only procure clients that perform name constraint checking in order to obtain assurance that unauthorized certificates are properly rejected. As an alternative, a federal agency may procure clients that use one or more of the features discussed in Appendix E.1.

The client **shall** terminate the TLS connection if path validation fails.

Federal agencies **shall** only use clients that check that the DNS name or IP address, whichever is presented in the client TLS request, matches a DNS name or IP address contained in the server certificate. The client **shall** terminate the TLS connection if the name check fails.

4.5.2 Trust Anchor Store

Having an excessive number of trust anchors installed in the TLS client can increase the chances for the client to be spoofed. As the number of trust anchors increase, the number of CAs that the client trusts increases, and the chances that one of these CAs or its registration system or process will be compromised to issue TLS server certificates also increases.

Clients **shall not** overpopulate their trust stores with various CA certificates that can be verified via cross-certification. Direct trust of these certificates can expose the clients unduly to a variety of situations, including but not limited to, revocation or compromise of these trust anchors. Direct trust also increases the operational and security burden on the clients to promulgate the addition and deletion of trust anchors. Instead, the client **shall** rely on the server overpopulating or not providing the hints list to mitigate the client certificate selection and path-building problem as discussed in Section 3.5.4.

4.5.3 Checking the Server Key Size

The only direct mechanism for a client to check if the key size presented in a server public certificate is acceptable is for the client to examine the server public key in the certificate. An indirect mechanism is to ensure that the server public-key certificate was issued under a policy that indicates the minimum cryptographic strength of the signature and hashing algorithms used. In some cases, this can be done by the client performing certificate policy processing and checking. However, since many TLS clients cannot be configured to accept or reject certificates based on the policies under which they were issued, this may require ensuring that the trust anchor store only contains CAs that issue certificates under acceptable policies. The client **shall** check the server public key length if the client implementation provides a mechanism to do so. The client **shall** also check the server public key length if the server uses ephemeral keys for the creation of the master secret, and the client implementation provides a mechanism to do so.

The length of each write key is determined by the negotiated cipher suite. Restrictions on the length of the shared session keys can be enforced by configuring the client to only support cipher suites that meet the key length requirements.

4.5.4 User Interface

When the TLS client is a browser, the browser interface can be used to determine if a TLS session is in effect. The indication that a TLS session is in effect varies by browser. Examples of indicators include a padlock in the URL bar, the word "secure" preceding the URL, or a different color for the URL bar. Some clients, such as browsers, may allow further investigation of the server certificate and negotiated session parameters by clicking on the lock (or other indicator). Users **should** examine the interface for the presence of the indicator to ensure that the TLS session is in force and **should** also visually examine web site URLs to ensure that the user intended to visit the indicated web site. Users **should** be aware that URLs can appear to be legitimate, but still not be valid. For example, the numeric "1" and the letter "l" appear quite similar or the same to the human eye.

Client authentication keys may be located outside of the client (e.g., PIV Cards). Users **shall** follow the relevant policies and procedures for protecting client authentication keys outside of the client.

4.6 Session Resumption and Early Data

Session resumption considerations and server recommendations were given in Section 3.6. There are no specific recommendations for clients regarding session resumption when using TLS 1.2, 1.1, or 1.0. Clients typically will not know if any anti-replay mechanisms are in place to prevent replay attacks on 0-RTT data in TLS 1.3. Therefore, clients using TLS 1.3 **should not** send 0-RTT data.

RFC 7918 [38] describes a technique, called False Start, that allows a TLS 1.2 client to send early data. While this concept is similar to the 0-RTT data of TLS 1.3, there are differences that affect security. For example, an attacker may perform downgrade attacks, both of protocol versions and cipher suites, and obtain client data before the handshake is determined to be invalid. While RFC 7918 provides recommendations for improving security, it is safest to disable False Start unless there is a real need for it. TLS 1.2 clients **shall not** use False Start.

4.7 Compression Methods

The client **shall** follow the same compression recommendations as the server, which are described in Section 3.7.

4.8 Operational Considerations

The client and associated platform **shall** be kept up-to-date in terms of security patches. This is critical to various aspects of security.

Once the TLS-protected data is received at the client, and decrypted and authenticated by the TLS layer of the client system, the unencrypted data is available to the applications on the client platform.

These guidelines do not mitigate the threats against the misuse or exposure of the client credentials that resides on the client machine. These credentials could contain the private key used for client authentication or other credentials (e.g., a one-time password (OTP) or user ID and password) for authenticating to a server-side application.

For these reasons, the use of TLS does not obviate the need for the client to use appropriate security measures, as described in applicable Federal Information Processing Standards and NIST Special Publications, to protect computer systems and applications. Users **shall** operate client systems in accordance with agency and administrator instructions.

Appendix A—Acronyms

Selected acronyms and abbreviations used in this paper are defined below.

3DES	Triple Data Encryption Algorithm (TDEA)
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
CA	Certification Authority
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CRL	Certificate Revocation List
DES	Data Encryption Standard
DH	Diffie-Hellman key exchange
DHE	Ephemeral Diffie-Hellman key exchange
DNS	Domain Name System
DNSSEC	DNS Security Extensions
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard (implies DSA)
EC	Elliptic Curve
ECDHE	Ephemeral Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standard
GCM	Galois Counter Mode
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
HMAC	Keyed-hash Message Authentication Code
IETF	Internet Engineering Task Force
KDF	Key derivation function
MAC	Message Authentication Code
OCSP	Online Certificate Status Protocol
OID	Object Identifier
PIV	Personal Identity Verification
PKI	Public Key Infrastructure
PRF	Pseudo-random Function
PSK	Pre-Shared Key

RFC	Request for Comments
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator

Appendix B—Interpreting Cipher Suite Names

TLS cipher suite names consist of a set of mnemonics separated by underscores (i.e., "_"). The naming convention in TLS 1.3 differs from the convention shared in TLS 1.0, 1.1, and 1.2. Section B.1 provides guidance for interpreting the names of cipher suites that are recommended in these guidelines for TLS versions 1.0, 1.1, and 1.2. Section B.2 provides guidance for interpreting the names of cipher suites for TLS 1.3. In all TLS cipher suites, the first mnemonic is the protocol name, i.e., "TLS".

B.1 Interpreting Cipher Suites Names in TLS 1.0, 1.1, and 1.2

As shown in Section 3.3.1, these cipher suites have the following form:

TLS_*KeyExchangeAlg*_WITH_*EncryptionAlg*_*MessageAuthenticationAlg*

KeyExchangeAlg consists of one or two mnemonics.

- If there is only one mnemonic, it must be PSK, based on the recommendations in these guidelines. The single mnemonic PSK indicates that the premaster secret is established using only symmetric algorithms with pre-shared keys, as described in RFC 4279 [29]. Pre-shared key cipher suites that are approved for use with TLS 1.2 are listed in Appendix C.
- If there are two mnemonics following the protocol name, the first key exchange mnemonic should be DH, ECDH, DHE, or ECDHE.
    - When the first key exchange mnemonic is DH or ECDH, it indicates that the server's public key in its certificate is for either DH or ECDH key exchange, and the second mnemonic indicates the signature algorithm that was used by the issuing CA to sign the server certificate.
    - When the first key exchange mnemonic is DHE or ECDHE, it indicates that ephemeral DH or ECDH will be used for key exchange, with the second mnemonic indicating the server signature public key type that will be used to authenticate the server's ephemeral public key.[33]

EncryptionAlg indicates the symmetric encryption algorithm and associated mode of operations.

MessageAuthenticationAlg is generally the hashing algorithm to be used for HMAC, if applicable.[34] In cases where HMAC is not applicable (e.g., AES-GCM), or the cipher suite was defined after the release of the TLS 1.2 RFC, this mnemonic represents the hashing algorithm used with the PRF.

The following examples illustrate how to interpret the cipher suite names:

[33] In this case, the signature algorithm used by the CA to sign the certificate is not articulated in the cipher suite.

[34] HMAC is not applicable when the symmetric encryption mode of operation is authenticated encryption. Note that the CCM mode cipher suites do not specify the last mnemonic and require that SHA-256 be used for the PRF.

- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256: Ephemeral DH is used for the key exchange. The server's ephemeral public key is authenticated using the server's RSA public key. Once the handshake is completed, the messages are encrypted using AES-256 in CBC mode. SHA-256 is used for both the PRF and HMAC computations.

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384: Ephemeral ECDH is used for key exchange. The server's ephemeral public key is authenticated using the server's ECDSA public key. Once the handshake is completed, the messages are encrypted and authenticated using AES-256 in GCM mode, and SHA-384 is used for the PRF. Since an authenticated encryption mode is used, messages neither have nor require an HMAC message authentication code.

B.2 Interpreting Cipher Suites Names in TLS 1.3

As shown in Section 3.3.1, these cipher suites have the following form:

TLS_*AEAD*_*HASH*

AEAD indicates the AEAD algorithm that is used for confidentiality, integrity, and message authentication. The NIST-approved TLS 1.3 AEAD algorithms comprise a NIST-recommended block cipher and NIST-recommended AEAD mode.

HASH indicates the hashing algorithm that is used as a pseudorandom function during key derivation.

The following examples illustrate how to interpret TLS 1.3 cipher suite names.

- TLS_AES_256_GCM_SHA384: messages are encrypted and authenticated with AES-256 in GCM mode, and SHA-384 is used with the HKDF.
- TLS_AES_128_CCM_SHA256: messages are encrypted and authenticated with AES-128 in CCM mode, and SHA-256 is used with the HKDF.

The negotiation of the key exchange method is handled elsewhere in the TLS handshake.

Appendix C—Pre-shared Keys

Pre-shared keys (PSK) are symmetric keys that are already in place prior to the initiation of a TLS session (e.g., as the result of a manual distribution). The use of PSKs in TLS versions prior to TLS 1.3 is described in RFC 4279 [29], RFC 5487 [3], and RFC 5489 [4]. Pre-shared keys are used for session resumption in TLS 1.3. In general, pre-shared keys **should not** be used in TLS versions prior to TLS 1.3, or for initial session establishment in TLS 1.3. However, the use of pre-shared keys may be appropriate for some closed environments that have adequate key management support. For example, they might be appropriate for constrained environments with limited processing, memory, or power. If PSKs are appropriate and supported, then the following additional guidelines **shall** be followed.

Recommended pre-shared key (PSK) cipher suites for TLS 1.2 are listed below. Cipher suites for TLS 1.3 (see Section 3.3.1.2) can all be used with pre-shared keys. Pre-shared keys **shall** be distributed in a secure manner, such as a secure manual distribution or using a key-establishment certificate. These cipher suites employ a pre-shared key for entity authentication (for both the server and the client) and may also use ephemeral Diffie-Hellman (DHE) or ephemeral Elliptic Curve Diffie-Hellman (ECDHE) algorithms for key establishment. For example, when DHE is used, the result of the Diffie-Hellman computation is combined with the pre-shared key and other input to determine the premaster secret.

The pre-shared key **shall** have a minimum security strength of 112 bits. Because these cipher suites require pre-shared keys, these suites are not generally applicable to common secure web site applications and are not expected to be widely supported in TLS clients or TLS servers. NIST suggests that these suites be considered for infrastructure applications, particularly if frequent authentication of the network entities is required.

Pre-shared key cipher suites may only be used in networks where both the client and server belong to the same organization. Cipher suites using pre-shared keys **shall not** be used with TLS 1.0 or TLS 1.1, and **shall not** be used when a government client or server communicates with non-government systems.

TLS 1.2 servers and clients using pre-shared keys may support the following cipher suites:

- TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 (0x00, 0xAA)
- TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 (0x00, 0xAB)
- TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256 (0xC0, 0x37)
- TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384 (0xC0, 0x38)
- TLS_DHE_PSK_WITH_AES_128_CCM (0xC0, 0xA6)
- TLS_DHE_PSK_WITH_AES_256_CCM (0xC0, 0xA7)
- TLS_PSK_DHE_WITH_AES_128_CCM_8 (0xC0, 0xAA)
- TLS_PSK_DHE_WITH_AES_256_CCM_8 (0xC0, 0xAB)
- TLS_DHE_PSK_WITH_AES_128_CBC_SHA256 (0x00, 0xB2)
- TLS_DHE_PSK_WITH_AES_256_CBC_SHA384 (0x00, 0xB3)
- TLS_PSK_WITH_AES_128_GCM_SHA256 (0x00, 0xA8)
- TLS_PSK_WITH_AES_256_GCM_SHA384 (0x00, 0xA9)
- TLS_PSK_WITH_AES_128_CCM (0xC0, 0xA4)

- TLS_PSK_WITH_AES_256_CCM (0xC0, 0xA5)
- TLS_PSK_WITH_AES_128_CCM_8 (0xC0, 0xA8)
- TLS_PSK_WITH_AES_256_CCM_8 (0xC0, 0xA9)
- TLS_PSK_WITH_AES_128_CBC_SHA256 (0x00, 0xAE)
- TLS_PSK_WITH_AES_256_CBC_SHA384 (0x00, 0xAF)
- TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA (0xC0, 0x35)
- TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA (0xC0, 0x36)
- TLS_DHE_PSK_WITH_AES_128_CBC_SHA (0x00, 0x90)
- TLS_DHE_PSK_WITH_AES_256_CBC_SHA (0x00, 0x91)
- TLS_PSK_WITH_AES_128_CBC_SHA (0x00, 0x8C)
- TLS_PSK_WITH_AES_256_CBC_SHA (0x00, 0x8D)

Appendix D—RSA Key Transport

RSA key transport is a key exchange mechanism where the premaster secret is chosen by the client, encrypted with the server's public key, and sent to the server. It is available in TLS versions 1.0 through 1.2, but it is not supported by TLS 1.3. While it is a convenient method for key exchange when the server's certificate contains an RSA public key, this method has several drawbacks:

1. The client has sole responsibility for the premaster secret generation. If the client does not have sufficient entropy to generate the secret, the security of the session will suffer.
2. It does not enable forward secrecy.
3. The padding scheme that TLS uses for this operation has a known vulnerability that requires TLS implementations to perform attack mitigation.

For these reasons, this guideline does not recommend cipher suites that use RSA key transport for key exchange (see Section 3.3.1).

Forward secrecy is often a security goal, as it prevents the compromise of long-term keys from enabling the decryption of sessions. The only way to achieve this property in TLS is to use a key exchange mechanism that relies on ephemeral parameters (i.e., cipher suites that contain DHE or ECDHE) as specified in RFC 5246 [24].

RSA key-transport using PKCS #1 v1.5 is vulnerable to vulnerable to Bleichenbacher oracle attacks. RFC 5246 contains steps to mitigate the attacks by processing incorrectly formatted messages in a manner indistinguishable from the processing of properly-formatted messages (see [24], Section 7.4.7.1). The mitigation techniques are not always effective in practice (for examples, see [13]).

D.1 Transition Period

While these guidelines do not recommend cipher suites using RSA key transport, there may be circumstances in practice where RSA key transport is needed. For example, if an agency uses a network appliance for regulatory or enterprise security purposes that only functions with these cipher suites, then these cipher suites may need to be enabled. It is recommended that agencies transition to a new method to meet their needs as soon as it is practical.

If RSA key transport is needed while a new traffic inspection strategy is being developed, only RSA key transport cipher suites from the following list may be used. See Section for 3.3.1.1 for general information on preference order.

- TLS_RSA_WITH_AES_128_CCM (xC0, x9C)
- TLS_RSA_WITH_AES_256_CCM (xC0, x9D)
- TLS_RSA_WITH_AES_128_CCM_8 (xC0, xA0)
- TLS_RSA_WITH_AES_256_CCM_8 (xC0, xA1)
- TLS_RSA_WITH_AES_128_CBC_SHA (x00, x2F)
- TLS_RSA_WITH_AES_256_CBC_SHA (x00, x35)
- TLS_RSA_WITH_AES_128_CBC_SHA256 (x00, 3C)

- TLS_RSA_WITH_AES_256_CBC_SHA256 (x00, 3D)
- TLS_RSA_WITH_AES_128_GCM_SHA256 (x00, x9C)
- TLS_RSA_WITH_AES_256_GCM_SHA384 (x00, x9D)

See transition guidance in SP 800-131A [10] for information on deprecation timelines.

Appendix E—Future Capabilities

This section identifies emerging concepts and capabilities that are applicable to TLS. As these concepts mature, and commercial products are available to support them, these guidelines will be revised to provide specific recommendations.

E.1 U.S. Federal Public Trust PKI

The Identity, Credential, and Access Management (ICAM) Subcommittee of the Federal CIO Council's Information Security and Identity Management Committee is developing a new public trust root and issuing CA infrastructure to issue TLS server certificates for federal web services on the public Internet. The intent is for this new root to be included in all of the commonly used trust stores so that federal agencies can obtain their TLS server certificates from this PKI rather than from commercial CAs. The certificate policy for this PKI is being developed at https://devicepki.idmanagement.gov.

Once this PKI is operational and is included in the commonly used trust stores, federal agencies should consider obtaining their TLS server certificates from this PKI.

E.2 DNS-based Authentication of Named Entities (DANE)

DANE leverages DNS security extensions (DNSSEC) to provide mechanisms for securely obtaining information about TLS server certificates from the DNS. RFC 6698 [33] specifies a resource record that may be made available in DNS that includes a certificate (or the public key of a certificate), along with an indicator of how the certificate is to be used. There are four options:

1. The DNS record contains an end-entity certificate. In addition to the server public-key certificate validation as specified in Section 4.5, the client verifies that the TLS server certificate matches the certificate provided in the DNS records.
2. The DNS record contains a domain-issued end-entity certificate.[35] The client can use the certificate if it verifies that the TLS server certificate matches the one provided in the DNS records (i.e., the client forgoes server public-key certificate validation as specified in Section 4.5).
3. The DNS record contains a CA certificate. In addition to the server public-key certificate validation as specified in Section 4.5, the client verifies that the certification path for the TLS server certificate includes the CA certificate provided in the DNS records.
4. The DNS record contains a certificate that is to be used as a trust anchor. The client validates the TLS server certificate as specified in Section 4.5 using the trust anchor provided in the DNS records instead of the trust anchors in the client's local trust anchor store.

[35] In this context, a "domain-issued" certificate is one that is issued by the domain name administrator without involving a third-party CA. It corresponds to usage case 3 in Section 2.1.1 of RFC 6698.

In each case, the client verifies the digital signatures on the DNS records in accordance with the DNSSEC, as described in RFC 4033 [2].

Appendix F—Determining the Need for TLS 1.0 and 1.1

Enabling TLS 1.0 or TLS 1.1 when they are not needed may leave systems and users vulnerable to attacks (such as the BEAST attack and the Klima attack [57]). However, disabling older versions of TLS when there is a need may deny access to users who are unable to install or upgrade to a client that is capable of TLS 1.3 or TLS 1.2.

The system administrator must consider the benefits and risks of using TLS 1.0 or TLS 1.1, in the context of applications supported by the server, and decide whether the benefits of using TLS 1.0 or TLS 1.1 outweigh the risks. This decision should be driven by the service(s) running on the server and the versions supported by clients accessing the server. Services that do not access high-value information (such as personally identifiable information or financial data) may benefit from using TLS 1.0 by increasing accessibility with little increased risk. On the other hand, services that do access high-value data may increase the likelihood of a breach for relatively little gain in terms of accessibility. The decision to support TLS 1.0 or TLS 1.1 must be technically assessed on a case-by-case basis. This is to ensure that supporting older TLS versions is absolutely necessary and that associated risks and business implications are understood and accepted.

These guidelines do not give specific recommendations on steps that can be taken to make this determination. There are tools available (such as the Data Analytics Program [68]) that can provide information to system administrators that can be used to assess the impact of supporting, or not supporting, TLS versions prior to TLS 1.2. For example, DAP data on visitor OS and browser versions can help administrators determine what percentage of visitors to agency websites cannot negotiate recommended TLS versions by default.

Many products that implement TLS 1.1 also implement TLS 1.2. Because of this, it may be unnecessary for servers to support TLS 1.1. Administrators can determine whether TLS 1.1 is needed by assessing whether it must support connections with clients where 1.1 is the highest TLS version available.

Appendix G—References

[1] AlFardan, N.J., and Paterson, K.G., *Lucky Thirteen: Breaking the TLS and DTLS Record Protocols*, February 2013, http://www.isg.rhul.ac.uk/tls/TLStiming.pdf

[2] Arends, R., Austein, R., Larson, M., Massey, D., and Rose, S., *DNS Security Introduction and Requirements*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 4033, March 2005, https://doi.org/10.17487/RFC4033

[3] Badra, M., *Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5487, March 2009, https://doi.org/10.17487/RFC5487

[4] Badra, M., and Hajjeh, I., *ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5489, March 2009, https://doi.org/10.17487/RFC5489

[5] Barker, E., *Recommendation for Key Management Part 1: General*, NIST Special Publication (SP) 800-57 Part 1 Revision 4, National Institute of Standards and Technology, Gaithersburg, Maryland, January 2016, https://doi.org/10.6028/NIST.SP.800-57pt1r4

[6] Barker, E., Chen, L., and Moody, D., *Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography*, NIST Special Publication (SP) 800-56B Revision 1, National Institute of Standards and Technology, Gaithersburg, Maryland September 2014, https://doi.org/10.6028/NIST.SP.800-56Br1

[7] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and Davis, R., *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, Special Publication (SP) 800-56A Revision 3, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2018, https://doi.org/10.6028/NIST.SP.800-56Ar3

[8] Barker, E., and Kelsey, J., *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, NIST Special Publication (SP) 800-90A Revision 1, National Institute of Standards and Technology, Gaithersburg, Maryland June 2015, https://doi.org/10.6028/NIST.SP.800-90Ar1

[9] Barker, E., and Mouha, N., *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, NIST Special Publication (SP) 800-67 Revision 2, National Institute of Standards and Technology, Gaithersburg, Maryland, November 2017, https://doi.org/10.6028/NIST.SP.800-67r2

[10] Barker, E., and Roginsky, A., *Transitioning the Use of Cryptographic Algorithms and Key Lengths*, NIST Special Publication (SP) 800-131A Revision 2 (draft), National Institute of Standards and Technology, Gaithersburg, Maryland July 2018, https://csrc.nist.gov/publications/detail/sp/800-131a/rev-2/draft

[11] Be'ery, T., and Shulman, A., *A Perfect CRIME? Only TIME Will Tell*, Blackhat Europe, 2013, https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-crime-beery-wp.pdf

[12] Bhargavan, K., Lavaud, A.D., Fournet, C., Pironti, A., and Strub, P.Y., *Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS*, 2014 IEEE Symposium on Security and Privacy, May 2014, pp. 98-113, https://doi.org/10.1109/SP.2014.14

[13] Böck, H., Somorovsky, J., and Young, C., *Return Of Bleichenbacher's Oracle Threat (ROBOT)*, Cryptology ePrint Archive, Report 2017/1189, 2017, https://eprint.iacr.org/2017/1189

[14] Bradner, S., *Key words for use in RFCs to Indicate Requirement Levels*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 2119, March 1997, https://doi.org/10.17487/RFC2119

[15] CA/Browser Forum, *Baseline Requirements Certificate Policy for the Issuance and Management of Publicly-Trusted Certificates*, https://cabforum.org/baseline-requirements-documents/

[16] CA/Browser Forum, *Guidelines For The Issuance And Management Of Extended Validation Certificates*, https://cabforum.org/extended-validation

[17] Chernick, C.M., III, C.E., Fanto, M.J., and Rosenthal, R., *Guidelines for the Selection and Use of Transport Layer Security (TLS) Implementations*, NIST Special Publication (SP) 800-52, National Institute of Standards and Technology, Gaithersburg, Maryland, June 2005, https://doi.org/10.6028/NIST.SP.800-52

[18] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and Polk, W., *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5280, 2008, https://doi.org/10.17487/RFC5280

[19] Dang, Q., *Recommendation for Applications Using Approved Hash Algorithms*, NIST Special Publication (SP) 800-107 Revision 1, National Institute of Standards and Technology, Gaithersburg, Maryland August 2012, https://doi.org/10.6028/NIST.SP.800-107r1

[20] Dang, Q., *Recommendation for Existing Application-Specific Key Derivation Functions*, NIST Special Publication (SP) 800-135 Revision 1, National Institute of Standards and Technology, Gaithersburg, Maryland, December 2011, https://doi.org/10.6028/NIST.SP.800-135r1

[21] Dang, Q., and Barker, E., *Recommendation for Key Management, Part 3: Application-Specific Key Management Guidance*, NIST Special Publication (SP) 800-57 Part 3 Revision 1, National Institute of Standards and Technology, Gaithersburg, Maryland, January 2015, https://doi.org/10.6028/NIST.SP.800-57pt3r1

[22] Dierks, T., and Allen, C., *The TLS Protocol Version 1.0*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 2246, January 1999, https://doi.org/10.17487/RFC2246

[23] Dierks, T., and Rescorla, E., *The Transport Layer Security (TLS) Protocol Version 1.1*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 4346, 2006, https://doi.org/10.17487/RFC4346

[24] Dierks, T., and Rescorla, E., *The Transport Layer Security (TLS) Protocol Version 1.2*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5246, August 2008, https://doi.org/10.17487/RFC5246

[25] Dworkin, M., *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, NIST Special Publication (SP) 800-38D, National Institute of Standards and Technology, Gaithersburg, Maryland, November 2007, https://doi.org/10.6028/NIST.SP.800-38D

[26] Dworkin, M., *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*, NIST Special Publication (SP) 800-38A, National Institute of Standards and Technology, Gaithersburg, Maryland, December 2001, https://doi.org/10.6028/NIST.SP.800-38A

[27] Dworkin, M., *Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*, NIST Special Publication (SP) 800-38C, National Institute of Standards and Technology, Gaithersburg, Maryland, May 2004, https://doi.org/10.6028/NIST.SP.800-38C

[28] Eastlake, D., 3rd, *Transport Layer Security (TLS) Extensions: Extension Definitions*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6066, January 2011, https://doi.org/10.17487/RFC6066

[29] Eronen, P., and Tschofenig, H., *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 4279, December 2005, https://doi.org/10.17487/RFC4279

[30] Federal Public Key Infrastructure Authority, *X.509 Certificate Policy For The U.S. Federal PKI Common Policy Framework*, https://www.idmanagement.gov/fpki/#certificate-policies

[31] Freier, A., Karlton, P., and Kocher, P., *The Secure Sockets Layer (SSL) Protocol Version 3.0*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6101, August 2011, https://doi.org/10.17487/RFC6101

[32] Gutmann, P., *Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 7366, September 2014, https://doi.org/10.17487/RFC7366

[33] Hoffman, P., and Schlyter, J., *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6698, August 2012, https://doi.org/10.17487/RFC6698

[34] Joint Task Force Transformation Initiative, *Security and Privacy Controls for Federal Information Systems and Organizations*, NIST Special Publication (SP) 800-53 Revision 4, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2013, https://doi.org/10.6028/NIST.SP.800-53r4

[35] K. Bhargavan, E., Delignat-Lavaud, A., Pironti, A., Langley, A., and Ray, M., *Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 7627, September 2015, https://doi.org/10.17487/RFC7627

[36] Krawczyk, H., and Eronen, P., *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5869, May 2010, https://doi.org/10.17487/RFC5869

[37] Langley, A., *The POODLE bites again*, https://www.imperialviolet.org/2014/12/08/poodleagain.html

[38] Langley, A., Modadugu, N., and Moeller, B., *Transport Layer Security (TLS) False Start*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 7918, August 2016, https://doi.org/10.17487/RFC7918

[39] Laurie, B., Langley, A., and Kasper, E., *Certificate Transparency*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6962, June 2013, https://doi.org/10.17487/RFC6962

[40] McGrew, D., and Bailey, D., *AES-CCM Cipher Suites for Transport Layer Security (TLS)*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6655, July 2012, https://doi.org/10.17487/RFC6655

[41] Moeller, B., and Langley, A., *TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 7507, April 2015, https://doi.org/10.17487/RFC7507

[42] Möller, B., Duong, T., and Kotowicz, K., *This POODLE Bites: Exploiting The SSL 3.0 Fallback*, September 2014, https://www.openssl.org/~bodo/ssl-poodle.pdf

[43] National Institute of Standards and Technology, *Random Bit Generation*, https://csrc.nist.gov/Projects/Random-Bit-Generation

[44] Nir, Y., Josefsson, S., and Pegourie-Gonnard, M., *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 8422, August 2018, https://doi.org/10.17487/RFC8422

[45] P. Wouters, E., H. Tschofenig, E., Gilmore, J., Weiler, S., and Kivinen, T., *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 7250, June 2014, https://doi.org/10.17487/RFC7250

[46] Paterson, K.G., Ristenpart, T., and Shrimpton, T., *Tag size does matter: attacks and proofs for the TLS record protocol*. Proc. 17th international conference on The Theory and Application of Cryptology and Information Security, Seoul, South Korea, 2011, Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information

Security, https://doi.org/10.1007/978-3-642-25385-0

[47] Pettersen, Y., *The Transport Layer Security (TLS) Multiple Certificate Status Request Extension*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6961, 2013, https://doi.org/10.17487/RFC6961

[48] Polk, T., McKay, K., and Chokhani, S., *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*, NIST Special Publication (SP) 800-52 Revision 1, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2014, https://doi.org/10.6028/NIST.SP.800-52r1

[49] Rescorla, E., *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5289, August 2008, https://doi.org/10.17487/RFC5289

[50] Rescorla, E., *The Transport Layer Security (TLS) Protocol Version 1.3*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 8446, August 2018, https://doi.org/10.17487/RFC8446

[51] Rescorla, E., Ray, M., Dispensa, S., and Oskov, N., *Transport Layer Security (TLS) Renegotiation Indication Extension*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5746, February 2010, https://doi.org/10.17487/RFC5746

[52] Rizzo, J., and Duong, T., *The CRIME Attack*, EKOparty Security Conference, 2012

[53] Salowey, J., Choudhury, A., and McGrew, D., *AES Galois Counter Mode (GCM) Cipher Suites for TLS*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5288, August 2008, https://doi.org/10.17487/RFC5288

[54] Salowey, J., Zhou, H., Eronen, P., and Tschofenig, H., *Transport Layer Security (TLS) Session Resumption without Server-Side State*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 5077, January 2008, https://doi.org/10.17487/RFC5077

[55] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C., *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6960, 2013, https://doi.org/10.17487/RFC6960

[56] Seggelmann, R., Tuexen, M., and Williams, M., *Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6520, February 2012, https://doi.org/10.17487/RFC6520

[57] Sheffer, Y., Holz, R., and Saint-Andre, P., *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 7457, February 2015, https://doi.org/10.17487/RFC7457

[58] Springall, D., Durumeric, Z., and Halderman, J.A., *Measuring the Security Harm of TLS*

Crypto Shortcuts. Proc. Proceedings of the 2016 Internet Measurement Conference, Santa Monica, California, USA, 2016, pp. 33-47, https://doi.org/10.1145/2987443.2987480

[59] The Federal Bridge Certification Authority, *X.509 Certificate Policy For The Federal Bridge Certification Authority (FBCA)*, https://www.idmanagement.gov/fpki/#certificate-policies

[60] Turan, M.S., Barker, E., Kelsey, J., McKay, K.A., Baish, M.L., and Boyle, M., *Recommendation for the Entropy Sources Used for Random Bit Generation*, NIST Special Publication (SP) 800-90B, National Institute of Standards and Technology, Gaithersburg, Maryland, January 2018, https://doi.org/10.6028/NIST.SP.800-90B

[61] U.S. Department of Commerce, *Advanced Encryption Standard*, Federal Information Processing Standards (FIPS) Publication 197, November 2001, https://doi.org/10.6028/NIST.FIPS.197

[62] U.S. Department of Commerce, *Digital Signature Standard (DSS)*, Federal Information Processing Standards (FIPS) Publication 186-4, July 2013, https://doi.org/10.6028/NIST.FIPS.186-4

[63] U.S. Department of Commerce, *The Keyed-Hash Message Authentication Code (HMAC)*, Federal Information Processing Standards (FIPS) Publication 198-1, July 2008, https://doi.org/10.6028/NIST.FIPS.198-1

[64] U.S. Department of Commerce, *Personal Identity Verification (PIV) of Federal Employees and Contractors*, Federal Information Processing Standards (FIPS) Publication 201-2, August 2013, https://doi.org/10.6028/NIST.FIPS.201-2

[65] U.S. Department of Commerce, *Secure Hash Standard (SHS)*, Federal Information Processing Standards (FIPS) Publication 180-4, August 2015, https://doi.org/10.6028/NIST.FIPS.180-4

[66] U.S. Department of Commerce, *Security Requirements for Cryptographic Modules*, Federal Information Processing Standards (FIPS) Publication 140-2, May 2001 (including Change Notice 2, December 3, 2002), https://doi.org/10.6028/NIST.FIPS.140-2

[67] U.S. Department of the Treasury, *Public Key Infrastructure (PKI) X.509 Certificate Policy, version 2.9*, March 2017, https://pki.treas.gov/docs/treasury_x509_certificate_policy.pdf

[68] U.S. General Services Administration, *DAP: Digital Analytics Program*, https://digital.gov/dap, [accessed September 24, 2018]

[69] US-CERT/NIST, *CVE-2014-0160*, National Vulnerability Database, 2014, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160

[70] Yee, P., *Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, Internet Engineering Task Force (IETF) Request for Comments (RFC) 6818, January 2013, https://doi.org/10.17487/RFC6818

Appendix H—Revision History

H.1 Original

The original version of SP 800-52 was published in June 2005 [17]. At the time, only TLS 1.0 was final (TLS 1.1 was still under development). TLS 1.1 became a standard in April 2006, and TLS 1.2 became a standard in August 2008. SP 800-52 became outdated, and guidance on keys and cipher suites was incorporated into SP 800-57 Part 3 [21]. In March 2013, SP 800-52 was withdrawn.

H.2 Revision 1

The first revision of SP 800-52 was published in April 2014 [48]. The revision was a new document that bore little resemblance to the original. At the time, TLS 1.2 was still not prevalent and the Federal PKI consisted mainly of RSA certificates. Recommendations were made with this in mind so that federal agencies could follow the guidelines with either existing technology or technology that was under development. Agencies were advised to develop a plan to migrate to TLS 1.2.

After revision 1 was posted, the guidance on keys and cipher suites was removed from SP 800-57 Part 3.

H.3 Revision 2

Since revision 1, support for TLS 1.2 and cipher suites using ephemeral key exchanges has increased, and new attacks have come to light. Revision 2 (this document) requires that TLS 1.2 be supported, and contains several changes to certificate and cipher suite recommendations.

Revision 2 includes recommendations for TLS 1.3. TLS 1.3 offers many improvements over previous versions of TLS, so revision 2 advises agencies to develop a plan to migrate to TLS 1.3.

Revision 2 also has increased discussion on TLS attacks and guidance on mitigation.

Certificate requirements have also changed in this revision. In particular, status information for TLS server certificates is required to be made available via the Online Certificate Status Protocol. This revision of the TLS guidelines relaxes requirements on which signature algorithms can sign which key types in certificates.