Transport Layer Security

Transport Layer Security (**TLS**) is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use in securing HTTPS remains the most publicly visible.

The TLS protocol aims primarily to provide security, including privacy (confidentiality), integrity, and authenticity through the use of cryptography, such as the use of certificates, between two or more communicating computer applications. It runs in the application layer and is itself composed of two layers: the TLS record and the TLS handshake protocols.

TLS is a proposed Internet Engineering Task Force (IETF) standard, first defined in 1999, and the current version is TLS 1.3, defined in August 2018. TLS builds on the now-deprecated **SSL (Secure Sockets Layer**) specifications (1994, 1995, 1996) developed by Netscape Communications for adding the HTTPS protocol to their Navigator web browser.

Description

Client-server applications use the TLS protocol to communicate across a network in a way designed to prevent eavesdropping and tampering.

Since applications can communicate either with or without TLS (or SSL), it is necessary for the client to request that the server set up a TLS connection.[1] One of the main ways of achieving this is to use a different port number for TLS connections. Port 80 is typically used for unencrypted HTTP traffic while port 443 is the common port used for encrypted HTTPS traffic. Another mechanism is to make a protocol-specific STARTTLS request to the server to switch the connection to TLS – for example, when using the mail and news protocols.

Once the client and server have agreed to use TLS, they negotiate a stateful connection by using a handshaking procedure (see § TLS handshake).[2] The protocols use a handshake with an asymmetric cipher to establish not only cipher settings but also a session-specific shared key with which further communication is encrypted using a symmetric cipher. During this handshake, the client and server agree on various parameters used to establish the connection's security:

- The handshake begins when a client connects to a TLS-enabled server requesting a secure connection and the client presents a list of supported cipher suites (ciphers and hash functions).
- From this list, the server picks a cipher and hash function that it also supports and notifies the client of the decision.
- The server usually then provides identification in the form of a digital certificate. The certificate contains the server name, the trusted certificate authority (CA) that vouches for the authenticity of the certificate, and the server's public encryption key.
- The client confirms the validity of the certificate before proceeding.
- To generate the session keys used for the secure connection, the client either:
    - encrypts a random number (*PreMasterSecret*) with the server's public key and sends the result to the server (which only the server should be able to decrypt with its private key); both parties then use the random number to generate a unique session key for subsequent encryption and decryption of data during the session, or
    - uses Diffie–Hellman key exchange to securely generate a random and unique session key for encryption and decryption that has the additional property of forward secrecy: if the server's private key is disclosed in future, it cannot be used to decrypt the current session, even if the session is intercepted and recorded by a third party.

This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the session key until the connection closes. If any one of the above steps fails, then the TLS handshake fails and the connection is not created.

TLS and SSL do not fit neatly into any single layer of the OSI model or the TCP/IP model.[3][4] TLS runs "on top of some reliable transport protocol (e.g., TCP),"[5] which would imply that it is above the transport layer. It serves encryption to higher layers, which is normally the function of the presentation layer. However, applications generally use TLS as if it were a transport layer,[3][4] even though applications using TLS must actively control initiating TLS handshakes and handling of exchanged authentication certificates.[5]

When secured by TLS, connections between a client (e.g., a web browser) and a server (e.g., wikipedia.org) should have one or more of the following properties:

- The connection is *private* (or *secure*) because a symmetric-key algorithm is used to encrypt the data transmitted. The keys for this symmetric encryption are generated uniquely for each connection and are based on a shared secret that was negotiated at the start of the session. The server and client negotiate the details of which encryption algorithm and cryptographic keys to use before the first byte of data is transmitted (see below). The negotiation of a shared secret is both secure (the negotiated secret is unavailable to eavesdroppers and cannot be obtained, even by an attacker who places themself in the middle of the connection) and reliable (no attacker can modify the communications during the negotiation without being detected).
- The identity of the communicating parties can be *authenticated* using public-key cryptography. This authentication is required for the server and optional for the client.[6]
- The connection is *reliable* because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.[7]:3

In addition to the above, careful configuration of TLS can provide additional privacy-related properties such as forward secrecy, ensuring that any future disclosure of encryption keys cannot be used to decrypt any TLS communications recorded in the past.

TLS supports many different methods for exchanging keys, encrypting data, and authenticating message integrity. As a result, secure configuration of TLS involves many configurable parameters, and not all choices provide all of the privacy-related properties described in the list above (see the tables below § Key exchange, § Cipher security, and § Data integrity).

Attempts have been made to subvert aspects of the communications security that TLS seeks to provide, and the protocol has been revised several times to address these security threats.

Developers of web browsers have repeatedly revised their products to defend against potential security weaknesses after these were discovered (see TLS/SSL support history of web browsers).

History and development

SSL and TLS protocols

Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011 (RFC 6176 (https://datatracker.ietf.org/doc/html/rfc6176))
SSL 3.0	1996	Deprecated in 2015 (RFC 7568 (https://datatracker.ietf.org/doc/html/rfc7568))
TLS 1.0	1999	Deprecated in 2021 (RFC 8996 (https://datatracker.ietf.org/doc/html/rfc8996))[8][9][10]
TLS 1.1	2006	Deprecated in 2021 (RFC 8996 (https://datatracker.ietf.org/doc/html/rfc8996))[8][9][10]
TLS 1.2	2008	In use since 2008[11][12]
TLS 1.3	2018	In use since 2018[12][13]

Secure Data Network System

The Transport Layer Security Protocol (TLS), together with several other basic network security platforms, was developed through a joint initiative begun in August 1986, among the National Security Agency, the National Bureau of Standards, the Defense Communications Agency, and twelve communications and computer corporations who initiated a special project called the Secure Data Network System (SDNS).[14] The program was described in September 1987 at the 10th National Computer Security Conference in an extensive set of published papers. The innovative research program focused on designing the next generation of secure computer communications network and product specifications to be implemented for applications on public and private internets. It was intended to complement the rapidly emerging new OSI internet standards moving forward both in the U.S. government's GOSIP Profiles and in the huge ITU-ISO JTC1 internet effort internationally. Originally known as the SP4 protocol, it was renamed TLS and subsequently published in 1995 as international standard ITU-T X.274|ISO/IEC 10736:1995.

Secure Network Programming

Early research efforts towards transport layer security included the Secure Network Programming (SNP) application programming interface (API), which in 1993 explored the approach of having a secure transport layer API closely resembling Berkeley sockets, to facilitate retrofitting pre-existing network applications with security measures.[15]

SSL 1.0, 2.0, and 3.0

Netscape developed the original SSL protocols, and Taher Elgamal, chief scientist at Netscape Communications from 1995 to 1998, has been described as the "father of SSL".[16][17][18][19] SSL version 1.0 was never publicly released because of serious security flaws in the protocol. Version 2.0, after being released in February 1995 was quickly discovered to contain a number of security and usability flaws. It used the same cryptographic keys for message authentication and encryption. It had a weak MAC construction that used the MD5 hash function with a secret prefix, making it vulnerable to length extension attacks. And it provided no protection for either the opening handshake or an explicit message close, both of which meant man-in-the-middle attacks could go undetected. Moreover, SSL 2.0 assumed a single service and a fixed domain certificate, conflicting with the widely used feature of virtual hosting in Web servers, so most websites were effectively impaired from using SSL.

These flaws necessitated the complete redesign of the protocol to SSL version 3.0.[20][18] Released in 1996, it was produced by Paul Kocher working with Netscape engineers Phil Karlton and Alan Freier, with a reference implementation by Christopher Allen and Tim Dierks of Consensus Development. Newer versions of SSL/TLS are based on SSL 3.0. The 1996 draft of SSL 3.0 was published by IETF as a historical document in RFC 6101 (https://datatracker.ietf.org/doc/html/rfc6101) .

SSL 2.0 was deprecated in 2011 by RFC 6176 (https://datatracker.ietf.org/doc/html/rfc6176) . In 2014, SSL 3.0 was found to be vulnerable to the POODLE attack that affects all block ciphers in SSL; RC4, the only non-block cipher supported by SSL 3.0, is also feasibly broken as used in SSL 3.0.[21] SSL 3.0 was deprecated in June 2015 by RFC 7568 (https://datatracker.ietf.org/doc/html/rfc7568) .

TLS 1.0

TLS 1.0 was first defined in RFC 2246 (https://datatracker.ietf.org/doc/html/rfc2246) in January 1999 as an upgrade of SSL Version 3.0, and written by Christopher Allen and Tim Dierks of Consensus Development. As stated in the RFC, "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough to preclude interoperability between TLS 1.0 and SSL 3.0". Tim Dierks later wrote that these changes, and the renaming from "SSL" to "TLS", were a face-saving gesture to Microsoft, "so it wouldn't look [like] the IETF was just rubberstamping Netscape's protocol".[22]

The PCI Council suggested that organizations migrate from TLS 1.0 to TLS 1.1 or higher before June 30, 2018.[23][24] In October 2018, Apple, Google, Microsoft, and Mozilla jointly announced they would deprecate TLS 1.0 and 1.1 in March 2020.[8]

TLS 1.1

TLS 1.1 was defined in RFC 4346 (https://datatracker.ietf.org/doc/html/rfc4346) in April 2006.[25] It is an update from TLS version 1.0. Significant differences in this version include:

- Added protection against cipher-block chaining (CBC) attacks.
  - The implicit initialization vector (IV) was replaced with an explicit IV.
  - Change in handling of padding errors.
- Support for IANA registration of parameters.[26]:2

Support for TLS versions 1.0 and 1.1 was widely deprecated by web sites around 2020, disabling access to Firefox versions before 24 and Chromium-based browsers before 29.[27][28][29]

TLS 1.2

TLS 1.2 was defined in RFC 5246 (https://datatracker.ietf.org/doc/html/rfc5246) in August 2008. It is based on the earlier TLS 1.1 specification. Major differences include:

- The MD5–SHA-1 combination in the pseudorandom function (PRF) was replaced with SHA-256, with an option to use cipher suite specified PRFs.
- The MD5–SHA-1 combination in the finished message hash was replaced with SHA-256, with an option to use cipher suite specific hash algorithms. However, the size of the hash in the

finished message must still be at least 96 bits.[30]

- The MD5–SHA-1 combination in the digitally signed element was replaced with a single hash negotiated during handshake, which defaults to SHA-1.
- Enhancement in the client's and server's ability to specify which hashes and signature algorithms they accept.
- Expansion of support for authenticated encryption ciphers, used mainly for Galois/Counter Mode (GCM) and CCM mode of Advanced Encryption Standard (AES) encryption.
- TLS Extensions definition and AES cipher suites were added.[26]:2

All TLS versions were further refined in RFC 6176 (https://datatracker.ietf.org/doc/html/rfc6176) in March 2011, removing their backward compatibility with SSL such that TLS sessions never negotiate the use of Secure Sockets Layer (SSL) version 2.0.

TLS 1.3

TLS 1.3 was defined in RFC 8446 (https://datatracker.ietf.org/doc/html/rfc8446) in August 2018. It is based on the earlier TLS 1.2 specification. Major differences from TLS 1.2 include:[31]

- Separating key agreement and authentication algorithms from the cipher suites
- Removing support for weak and less-used named elliptic curves
- Removing support for MD5 and SHA-224 cryptographic hash functions
- Requiring digital signatures even when a previous configuration is used
- Integrating HKDF and the semi-ephemeral DH proposal
- Replacing resumption with PSK and tickets
- Supporting 1-RTT handshakes and initial support for 0-RTT
- Mandating perfect forward secrecy, by means of using ephemeral keys during the (EC)DH key agreement
- Dropping support for many insecure or obsolete features including compression, renegotiation, non-AEAD ciphers, non-PFS key exchange (among which are static RSA and static DH key exchanges), custom DHE groups, EC point format negotiation, Change Cipher Spec protocol, Hello message UNIX time, and the length field AD input to AEAD ciphers
- Prohibiting SSL or RC4 negotiation for backwards compatibility

- Integrating use of session hash
- Deprecating use of the record layer version number and freezing the number for improved backwards compatibility
- Moving some security-related algorithm details from an appendix to the specification and relegating ClientKeyShare to an appendix
- Adding the ChaCha20 stream cipher with the Poly1305 message authentication code
- Adding the Ed25519 and Ed448 digital signature algorithms
- Adding the x25519 and x448 key exchange protocols
- Adding support for sending multiple OCSP responses
- Encrypting all handshake messages after the ServerHello

Network Security Services (NSS), the cryptography library developed by Mozilla and used by its web browser Firefox, enabled TLS 1.3 by default in February 2017.[32] TLS 1.3 support was subsequently added — but due to compatibility issues for a small number of users, not automatically enabled[33] — to Firefox 52.0, which was released in March 2017. TLS 1.3 was enabled by default in May 2018 with the release of Firefox 60.0.[34]

Google Chrome set TLS 1.3 as the default version for a short time in 2017. It then removed it as the default, due to incompatible middleboxes such as Blue Coat web proxies.[35]

During the IETF 100 Hackathon, which took place in Singapore in 2017, the TLS Group worked on adapting open-source applications to use TLS 1.3.[36][37] The TLS group was made up of individuals from Japan, United Kingdom, and Mauritius via the cyberstorm.mu team.[37] This work was continued in the IETF 101 Hackathon in London,[38] and the IETF 102 Hackathon in Montreal.[39]

wolfSSL enabled the use of TLS 1.3 as of version 3.11.1, released in May 2017.[40] As the first commercial TLS 1.3 implementation, wolfSSL 3.11.1 supported Draft 18 and now supports Draft 28,[41] the final version, as well as many older versions. A series of blogs were published on the performance difference between TLS 1.2 and 1.3.[42]

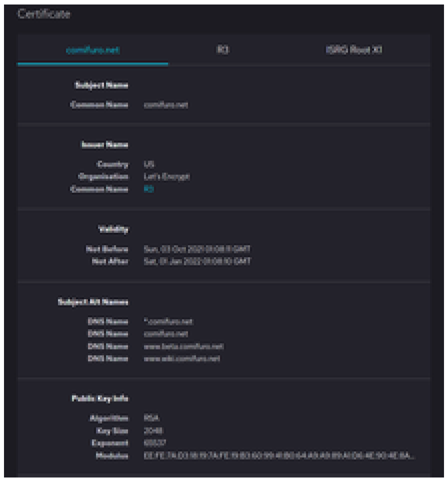
In September 2018, the popular OpenSSL project released version 1.1.1 of its library, in which support for TLS 1.3 was "the headline new feature".[43]

Support for TLS 1.3 was first added to Schannel with Windows 11 and Windows Server 2022.[44]

Enterprise Transport Security

The Electronic Frontier Foundation praised TLS 1.3 and expressed concern about the variant protocol Enterprise Transport Security (ETS) that intentionally disables important security measures in TLS 1.3.[45] Originally called Enterprise TLS (eTLS), ETS is a published standard known as the 'ETSI TS103523-3', "Middlebox Security Protocol, Part3: Enterprise Transport Security". It is intended for use entirely within proprietary networks such as banking systems. ETS does not support forward secrecy so as to allow third-party organizations connected to the proprietary networks to be able to use their private key to monitor network traffic for the detection of malware and to make it easier to conduct audits.[46][47] Despite the claimed benefits, the EFF warned that the loss of forward secrecy could make it easier for data to be exposed along with saying that there are better ways to analyze traffic.

Digital certificates



```
         Subject Name
          Common Name   comifuro.net

          Issuer Name
              Country   US
         Organisation   Let's Encrypt
          Common Name   R3

             Validity
           Not Before   Sun, 03 Oct 2021 01:08:11 GMT
            Not After   Sat, 01 Jan 2022 01:08:10 GMT

    Subject Alt Names
             DNS Name   *.comifuro.net
             DNS Name   comifuro.net
             DNS Name   www.beta.comifuro.net
             DNS Name   www.wiki.comifuro.net

      Public Key Info
            Algorithm   RSA
             Key Size   2048
             Exponent   65537
              Modulus   EE:FE:7A:D3:18:19:7A:FE:19:B3:60:99:41:B0:64:A9:A9:89:A1:D6:4E:90:4E:8A...
```

Example of a website with digital certificate

A digital certificate certifies the ownership of a public key by the named subject of the certificate, and indicates certain expected usages of that key. This allows others (relying parties) to rely upon signatures or on assertions made by the private key that corresponds to the certified public key. Keystores and trust stores can be in various formats, such as .pem, .crt, .pfx, and .jks.

Certificate authorities

TLS typically relies on a set of trusted third-party certificate authorities to establish the authenticity of certificates. Trust is usually anchored in a list of certificates distributed with user agent software,[48] and can be modified by the relying party.

According to Netcraft, who monitors active TLS certificates, the market-leading certificate authority (CA) has been Symantec since the beginning of their survey (or VeriSign before the authentication services business unit was purchased by Symantec). As of 2015, Symantec accounted for just under a third of all certificates and 44% of the valid certificates used by the 1 million busiest websites, as counted by Netcraft.[49] In 2017, Symantec sold its TLS/SSL business to DigiCert.[50] In an updated report, it was shown that IdenTrust, DigiCert, and Sectigo are the top 3 certificate authorities in terms of market share since May 2019.[51]

As a consequence of choosing X.509 certificates, certificate authorities and a public key infrastructure are necessary to verify the relation between a certificate and its owner, as well as to generate, sign, and administer the validity of certificates. While this can be more convenient than verifying the identities via a web of trust, the 2013 mass surveillance disclosures made it more widely known that certificate authorities are a weak point from a security standpoint, allowing man-in-the-middle attacks (MITM) if the certificate authority cooperates (or is compromised).[52][53]

Algorithms

Key exchange or key agreement

Before a client and server can begin to exchange information protected by TLS, they must securely exchange or agree upon an encryption key and a cipher to use when encrypting data (see § Cipher). Among the methods used for key exchange/agreement are: public and private keys generated with RSA (denoted TLS_RSA in the TLS handshake protocol), Diffie–Hellman (TLS_DH), ephemeral Diffie–Hellman (TLS_DHE), elliptic-curve Diffie–Hellman (TLS_ECDH), ephemeral elliptic-curve Diffie–Hellman (TLS_ECDHE), anonymous Diffie–Hellman (TLS_DH_anon),[7] pre-shared key (TLS_PSK)[54] and Secure Remote Password (TLS_SRP).[55]

The TLS_DH_anon and TLS_ECDH_anon key agreement methods do not authenticate the server or the user and hence are rarely used because those are vulnerable to man-in-the-middle attacks. Only TLS_DHE and TLS_ECDHE provide forward secrecy.

Public key certificates used during exchange/agreement also vary in the size of the public/private encryption keys used during the exchange and hence the robustness of the security provided. In July 2013, Google announced that it would no longer use 1024-bit public keys and would switch instead to 2048-bit keys to increase the security of the TLS encryption it provides to its users because the encryption strength is directly related to the key size.[56][57]

Key exchange/agreement and authentication

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
RSA	Yes	Yes	Yes	Yes	Yes	No	Defined for TLS 1.2 in RFCs
DH-RSA	No	Yes	Yes	Yes	Yes	No	
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes	
ECDH-RSA	No	No	Yes	Yes	Yes	No	
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
DH-DSS	No	Yes	Yes	Yes	Yes	No	
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[58]	
ECDH-ECDSA	No	No	Yes	Yes	Yes	No	
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDH-EdDSA	No	No	Yes	Yes	Yes	No	
ECDHE-EdDSA (forward secrecy)[59]	No	No	Yes	Yes	Yes	Yes	
PSK	No	No	Yes	Yes	Yes	?	
PSK-RSA	No	No	Yes	Yes	Yes	?	
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
SRP	No	No	Yes	Yes	Yes	?	
SRP-DSS	No	No	Yes	Yes	Yes	?	
SRP-RSA	No	No	Yes	Yes	Yes	?	
Kerberos	No	No	Yes	Yes	Yes	?	
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes	?	
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes	?	
GOST R 34.10-94/34.10-2001[60]	No	No	Yes	Yes	Yes	?	Proposed in RFC drafts

Cipher

Cipher security against publicly known feasible attacks

Cipher			Protocol version			
Type	Algorithm	Nominal strength (bits)	SSL 2.0	SSL 3.0[n 1][n 2][n 3][n 4]	TLS 1.0[n 1][n 3]	TLS 1.1
Block cipher with mode of operation	AES GCM[61][n 5]	256, 128	—	—	—	—
	AES CCM[62][n 5]		—	—	—	—
	AES CBC[n 6]		—	Insecure	Depends on mitigations	Depend on mitigatic
	Camellia GCM[63][n 5]	256, 128	—	—	—	—
	Camellia CBC[64][n 6]		—	Insecure	Depends on mitigations	Depend on mitigatic
	ARIA GCM[65][n 5]	256, 128	—	—	—	—
	ARIA CBC[65][n 6]		—	—	Depends on mitigations	Depend on mitigatic
	SEED CBC[66][n 6]	128	—	Insecure	Depends on mitigations	Depend on mitigatic
	3DES EDE CBC[n 6][n 7]	112[n 8]	Insecure	Insecure	Insecure	Insecu
	GOST 28147-89 CNT[60][n 7]	256	—	—	Insecure	Insecu
	IDEA CBC[n 6][n 7][n 9]	128	Insecure	Insecure	Insecure	Insecu
	DES CBC[n 6][n 7][n 9]	56	Insecure	Insecure	Insecure	Insecu

		40[n 10]	Insecure	Insecure	Insecure	—
	RC2 CBC[n 6][n 7]	40[n 10]	Insecure	Insecure	Insecure	—
Stream cipher	ChaCha20-Poly1305[71][n 5]	256	—	—	—	—
	RC4[n 11]	128	Insecure	Insecure	Insecure	Insecu
		40[n 10]	Insecure	Insecure	Insecure	—
None	**Null**[n 12]	–	Insecure	Insecure	Insecure	Insecu

Notes

1. *RFC 5746 (https://datatracker.ietf.org/doc/html/rfc5746) must be implemented to fix a renegotiation flaw that would otherwise break this protocol.*
2. *If libraries implement fixes listed in RFC 5746 (https://datatracker.ietf.org/doc/html/rfc5746) , this violates the SSL 3.0 specification, which the IETF cannot change unlike TLS. Most current libraries implement the fix and disregard the violation that this causes.*
3. *The BEAST attack breaks all block ciphers (CBC ciphers) used in SSL 3.0 and TLS 1.0 unless mitigated by the client and/or the server. See § Web browsers.*
4. *The POODLE attack breaks all block ciphers (CBC ciphers) used in SSL 3.0 unless mitigated by the client and/or the server. See § Web browsers.*
5. *AEAD ciphers (such as GCM and CCM) can only be used in TLS 1.2 or later.*

6. *CBC ciphers can be attacked with the Lucky Thirteen attack if the library is not written carefully to eliminate timing side channels.*
7. *The Sweet32 attack breaks block ciphers with a block size of 64 bits.*[67]
8. *Although the key length of 3DES is 168 bits, effective security strength of 3DES is only 112 bits,*[68] *which is below the recommended minimum of 128 bits.*[69]
9. *IDEA and DES have been removed from TLS 1.2.*[70]
10. *40-bit strength cipher suites were intentionally designed with reduced key lengths to comply with since-rescinded US regulations forbidding the export of cryptographic software containing certain strong encryption algorithms (see Export of cryptography from the United States). These weak suites are forbidden in TLS 1.1 and later.*
11. *Use of RC4 in all versions of TLS is prohibited by RFC 7465 (https://datatracker.ietf.org/doc/html/rfc7465) (because RC4 attacks weaken or break RC4 used in SSL/TLS).*
12. *Authentication only, no encryption.*

Data integrity

A message authentication code (MAC) is used for data integrity. HMAC is used for CBC mode of block ciphers. Authenticated encryption (AEAD) such as GCM and CCM mode uses AEAD-integrated MAC and doesn't use HMAC.[72] HMAC-based PRF, or HKDF is used for TLS handshake.

Data integrity

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
HMAC-MD5	Yes	Yes	Yes	Yes	Yes	No	Defined for TLS 1.2 in RFCs
HMAC-SHA1	No	Yes	Yes	Yes	Yes	No	
HMAC-SHA256/384	No	No	No	No	Yes	No	
AEAD	No	No	No	No	Yes	Yes	
GOST 28147-89 IMIT[60]	No	No	Yes	Yes	Yes	?	Proposed in RFC drafts
GOST R 34.11-94[60]	No	No	Yes	Yes	Yes	?	

Applications and adoption

In applications design, TLS is usually implemented on top of Transport Layer protocols, encrypting all of the protocol-related data of protocols such as HTTP, FTP, SMTP, NNTP and XMPP.

Historically, TLS has been used primarily with reliable transport protocols such as the Transmission Control Protocol (TCP). However, it has also been implemented with datagram-oriented transport protocols, such as the User Datagram Protocol (UDP) and the Datagram Congestion Control Protocol (DCCP), usage of which has been standardized independently using the term Datagram Transport Layer Security (DTLS).

Websites

A primary use of TLS is to secure World Wide Web traffic between a website and a web browser encoded with the HTTP protocol. This use of TLS to secure HTTP traffic constitutes the HTTPS protocol.[73]

Website protocol support (May 2022)

Protocol version	Website support[74]	Security[74][75]
SSL 2.0	0.3%	Insecure
SSL 3.0	2.5%	Insecure[76]
TLS 1.0	37.1%	Deprecated[8][9][10]
TLS 1.1	40.6%	Deprecated[8][9][10]
TLS 1.2	99.7%	Depends on cipher[n 1] and client mitigations[n 2]
TLS 1.3	54.2%	Secure

Notes

1. *see § Cipher table above*
2. *see § Web browsers and § Attacks against TLS/SSL sections*

Web browsers

As of April 2016, the latest versions of all major web browsers support TLS 1.0, 1.1, and 1.2, and have them enabled by default. However, not all supported Microsoft operating systems support the latest version of IE. Additionally, many Microsoft operating systems currently support multiple versions of IE, but this has changed according to Microsoft's Internet Explorer Support

Lifecycle Policy FAQ (https://support.microsoft.com/en-us/gp/microsoft-internet-explorer) , "beginning January 12, 2016, only the most current version of Internet Explorer available for a supported operating system will receive technical support and security updates." The page then goes on to list the latest supported version of IE at that date for each operating system. The next critical date would be when an operating system reaches the end of life stage. Since June 15, 2022, Internet Explorer 11 dropped support for Windows 10 editions which follow Microsoft's Modern Lifecycle Policy.[77][78]

Mitigations against known attacks are not enough yet:

- Mitigations against POODLE attack: some browsers already prevent fallback to SSL 3.0; however, this mitigation needs to be supported by not only clients but also servers. Disabling SSL 3.0 itself, implementation of "anti-POODLE record splitting", or denying CBC ciphers in SSL 3.0 is required.
    - Google Chrome: complete (TLS_FALLBACK_SCSV is implemented since version 33, fallback to SSL 3.0 is disabled since version 39, SSL 3.0 itself is disabled by default since version 40. Support of SSL 3.0 itself was dropped since version 44.)
    - Mozilla Firefox: complete (support of SSL 3.0 itself is dropped since version 39. SSL 3.0 itself is disabled by default and fallback to SSL 3.0 are disabled since version 34, TLS_FALLBACK_SCSV is implemented since version 35. In ESR, SSL 3.0 itself is disabled by default and TLS_FALLBACK_SCSV is implemented since ESR 31.3.0.)
    - Internet Explorer: partial (only in version 11, SSL 3.0 is disabled by default since April 2015. Version 10 and older are still vulnerable against POODLE.)
    - Opera: complete (TLS_FALLBACK_SCSV is implemented since version 20, "anti-POODLE record splitting", which is effective only with client-side implementation, is implemented since version 25, SSL 3.0 itself is disabled by default since version 27. Support of SSL 3.0 itself will be dropped since version 31.)
    - Safari: complete (only on OS X 10.8 and later and iOS 8, CBC ciphers during fallback to SSL 3.0 is denied, but this means it will use RC4, which is not recommended as well. Support of SSL 3.0 itself is dropped on OS X 10.11 and later and iOS 9.)
- Mitigation against RC4 attacks:
    - Google Chrome disabled RC4 except as a fallback since version 43. RC4 is disabled since Chrome 48.

    - Firefox disabled RC4 except as a fallback since version 36. Firefox 44 disabled RC4 by default.
    - Opera disabled RC4 except as a fallback since version 30. RC4 is disabled since Opera 35.
    - Internet Explorer for Windows 7/Server 2008 R2 and for Windows 8/Server 2012 have set the priority of RC4 to lowest and can also disable RC4 except as a fallback through registry settings. Internet Explorer 11 Mobile 11 for Windows Phone 8.1 disable RC4 except as a fallback if no other enabled algorithm works. Edge and IE 11 disable RC4 completely in August 2016.
- Mitigation against FREAK attack:
    - The Android Browser included with Android 4.0 and older is still vulnerable to the FREAK attack.
    - Internet Explorer 11 Mobile is still vulnerable to the FREAK attack.
    - Google Chrome, Internet Explorer (desktop), Safari (desktop & mobile), and Opera (mobile) have FREAK mitigations in place.
    - Mozilla Firefox on all platforms and Google Chrome on Windows were not affected by FREAK.

Libraries

Most SSL and TLS programming libraries are free and open source software.

- BoringSSL, a fork of OpenSSL for Chrome/Chromium and Android as well as other Google applications.
- Botan, a BSD-licensed cryptographic library written in C++.
- BSAFE Micro Edition Suite: a multi-platform implementation of TLS written in C using a FIPS-validated cryptographic module
- BSAFE SSL-J: a TLS library providing both a proprietary API and JSSE API, using FIPS-validated cryptographic module
- cryptlib: a portable open source cryptography library (includes TLS/SSL implementation)
- Delphi programmers may use a library called Indy which utilizes OpenSSL or alternatively ICS which supports TLS 1.3 now.
- GnuTLS: a free implementation (LGPL licensed)

- Java Secure Socket Extension (JSSE): the Java API and provider implementation (named SunJSSE)[79]
- LibreSSL: a fork of OpenSSL by OpenBSD project.
- MatrixSSL: a dual licensed implementation
- Mbed TLS (previously PolarSSL): A tiny SSL library implementation for embedded devices that is designed for ease of use
- Network Security Services: FIPS 140 validated open source library
- OpenSSL: a free implementation (BSD license with some extensions)
- Schannel: an implementation of SSL and TLS Microsoft Windows as part of its package.
- Secure Transport: an implementation of SSL and TLS used in OS X and iOS as part of their packages.
- wolfSSL (previously CyaSSL): Embedded SSL/TLS Library with a strong focus on speed and size.

A paper presented at the 2012 ACM conference on computer and communications security[80] showed that many applications used some of these SSL libraries incorrectly, leading to vulnerabilities. According to the authors:

> "The root cause of most of these vulnerabilities is the terrible design of the APIs to the underlying SSL libraries. Instead of expressing high-level security properties of network tunnels such as confidentiality and authentication, these APIs expose low-level details of the SSL protocol to application developers. As a consequence, developers often use SSL APIs incorrectly, misinterpreting and misunderstanding their manifold parameters, options, side effects, and return values."

Other uses

The Simple Mail Transfer Protocol (SMTP) can also be protected by TLS. These applications use public key certificates to verify the identity of endpoints.

TLS can also be used for tunnelling an entire network stack to create a VPN, which is the case with OpenVPN and OpenConnect. Many vendors have by now married TLS's encryption and authentication capabilities with authorization. There has also been substantial development since the late 1990s in creating client technology outside of Web-browsers, in order to enable support for client/server applications. Compared to traditional IPsec VPN technologies, TLS has some inherent advantages in firewall and NAT traversal that make it easier to administer for large remote-access populations.

TLS is also a standard method for protecting Session Initiation Protocol (SIP) application signaling. TLS can be used for providing authentication and encryption of the SIP signalling associated with VoIP and other SIP-based applications.[81]

Security

Attacks against TLS/SSL

Significant attacks against TLS/SSL are listed below.

In February 2015, IETF issued an informational RFC[82] summarizing the various known attacks against TLS/SSL.

Renegotiation attack

A vulnerability of the renegotiation procedure was discovered in August 2009 that can lead to plaintext injection attacks against SSL 3.0 and all current versions of TLS.[83] For example, it allows an attacker who can hijack an https connection to splice their own requests into the beginning of the conversation the client has with the web server. The attacker can't actually decrypt the client–server communication, so it is different from a typical man-in-the-middle attack. A short-term fix is for web servers to stop allowing renegotiation, which typically will not require other changes unless client certificate authentication is used. To fix the vulnerability, a renegotiation indication extension was proposed for TLS. It will require the client and server to include and verify information about previous handshakes in any renegotiation handshakes.[84] This extension has become a proposed standard and has been assigned the number RFC 5746 (https://datatracker.ietf.org/doc/html/rfc5746) . The RFC has been implemented by several libraries.[85][86][87]

Downgrade attacks: FREAK attack and Logjam attack

A protocol downgrade attack (also called a version rollback attack) tricks a web server into negotiating connections with previous versions of TLS (such as SSLv2) that have long since been abandoned as insecure.

Previous modifications to the original protocols, like **False Start**[88] (adopted and enabled by Google Chrome[89]) or **Snap Start**, reportedly introduced limited TLS protocol downgrade attacks[90] or allowed modifications to the cipher suite list sent by the client to the server. In doing so, an attacker might succeed in influencing the cipher suite selection in an attempt to downgrade the cipher suite negotiated to use either a weaker symmetric encryption algorithm or a weaker key exchange.[91] A paper presented at an ACM conference on computer and communications security in 2012 demonstrated that the False Start extension was at risk: in certain circumstances it could allow an attacker to recover the encryption keys offline and to access the encrypted data.[92]

Encryption downgrade attacks can force servers and clients to negotiate a connection using cryptographically weak keys. In 2014, a man-in-the-middle attack called FREAK was discovered affecting the OpenSSL stack, the default Android web browser, and some Safari browsers.[93] The attack involved tricking servers into negotiating a TLS connection using cryptographically weak 512 bit encryption keys.

Logjam is a security exploit discovered in May 2015 that exploits the option of using legacy "export-grade" 512-bit Diffie–Hellman groups dating back to the 1990s.[94] It forces susceptible servers to downgrade to cryptographically weak 512-bit Diffie–Hellman groups. An attacker can then deduce the keys the client and server determine using the Diffie–Hellman key exchange.

Cross-protocol attacks: DROWN

The DROWN attack is an exploit that attacks servers supporting contemporary SSL/TLS protocol suites by exploiting their support for the obsolete, insecure, SSLv2 protocol to leverage an attack on connections using up-to-date protocols that would otherwise be secure.[95][96] DROWN exploits a vulnerability in the protocols used and the configuration of the server, rather than any specific implementation error. Full details of DROWN were announced in March 2016, together with a patch for the exploit. At that time, more than 81,000 of the top 1 million most popular websites were among the TLS protected websites that were vulnerable to the DROWN attack.[96]

BEAST attack

On September 23, 2011 researchers Thai Duong and Juliano Rizzo demonstrated a proof of concept called **BEAST** (**Browser Exploit Against SSL/TLS**)[97] using a Java applet to violate same origin policy constraints, for a long-known cipher block chaining (CBC) vulnerability in TLS 1.0:[98][99] an attacker observing 2 consecutive ciphertext blocks C0, C1 can test if the plaintext block P1 is equal to x by choosing the next plaintext block $P2 = x \oplus C0 \oplus C1$; as per CBC operation, $C2 = E(C1 \oplus P2) = E(C1 \oplus x \oplus C0 \oplus C1) = E(C0 \oplus x)$, which will be equal to C1 if x = P1. Practical exploits had not been previously demonstrated for this vulnerability, which was originally discovered by Phillip Rogaway[100] in 2002. The vulnerability of the attack had been fixed with TLS 1.1 in 2006, but TLS 1.1 had not seen wide adoption prior to this attack demonstration.

RC4 as a stream cipher is immune to BEAST attack. Therefore, RC4 was widely used as a way to mitigate BEAST attack on the server side. However, in 2013, researchers found more weaknesses in RC4. Thereafter enabling RC4 on server side was no longer recommended.[101]

Chrome and Firefox themselves are not vulnerable to BEAST attack,[102][103] however, Mozilla updated their NSS libraries to mitigate BEAST-like attacks. NSS is used by Mozilla Firefox and Google Chrome to implement SSL. Some web servers that have a broken implementation of the SSL specification may stop working as a result.[104]

Microsoft released Security Bulletin MS12-006 on January 10, 2012, which fixed the BEAST vulnerability by changing the way that the Windows Secure Channel (Schannel) component transmits encrypted network packets from the server end.[105] Users of Internet Explorer (prior to version 11) that run on older versions of Windows (Windows 7, Windows 8 and Windows Server 2008 R2) can restrict use of TLS to 1.1 or higher.

Apple fixed BEAST vulnerability by implementing 1/n-1 split and turning it on by default in OS X Mavericks, released on October 22, 2013.[106]

CRIME and BREACH attacks

The authors of the BEAST attack are also the creators of the later CRIME attack, which can allow an attacker to recover the content of web cookies when data compression is used along with TLS.[107][108] When used to recover the content of secret authentication cookies, it allows an attacker to perform session hijacking on an authenticated web session.

While the CRIME attack was presented as a general attack that could work effectively against a large number of protocols, including but not limited to TLS, and application-layer protocols such as SPDY or HTTP, only exploits against TLS and SPDY were demonstrated and largely mitigated in browsers and servers. The CRIME exploit against HTTP compression has not been mitigated at all, even though the authors of CRIME have warned that this vulnerability might be even more widespread than SPDY and TLS compression combined. In 2013 a new instance of the CRIME attack against HTTP compression, dubbed BREACH, was announced. Based on the CRIME attack a BREACH attack can extract login tokens, email addresses or other sensitive information from TLS encrypted web traffic in as little as 30 seconds (depending on the number of bytes to be extracted), provided the attacker tricks the victim into visiting a malicious web link or is able to inject content into valid pages the user is visiting (ex: a wireless network under the control of the attacker).[109] All versions of TLS and SSL are at risk from BREACH regardless of the encryption algorithm or cipher used.[110] Unlike previous instances of CRIME, which can be successfully defended against by turning off TLS compression or SPDY header compression, BREACH exploits HTTP compression which cannot realistically be turned off, as virtually all web servers rely upon it to improve data transmission speeds for users.[109] This is a known limitation of TLS as it is susceptible to chosen-plaintext attack against the application-layer data it was meant to protect.

Timing attacks on padding

Earlier TLS versions were vulnerable against the padding oracle attack discovered in 2002. A novel variant, called the Lucky Thirteen attack, was published in 2013.

Some experts[69] also recommended avoiding triple DES CBC. Since the last supported ciphers developed to support any program using Windows XP's SSL/TLS library like Internet Explorer on Windows XP are RC4 and Triple-DES, and since RC4 is now deprecated (see discussion of RC4 attacks), this makes it difficult to support any version of SSL for any program using this library on XP.

A fix was released as the Encrypt-then-MAC extension to the TLS specification, released as RFC 7366 (https://datatracker.ietf.org/doc/html/rfc7366) .[111] The Lucky Thirteen attack can be mitigated in TLS 1.2 by using only AES_GCM ciphers; AES_CBC remains vulnerable.

POODLE attack

On October 14, 2014, Google researchers published a vulnerability in the design of SSL 3.0, which makes CBC mode of operation with SSL 3.0 vulnerable to a padding attack (CVE-2014-3566 (https://www.cve.org/CVERecord?id=CVE-2014-3566)). They named this attack **POODLE** (**Padding Oracle On Downgraded Legacy Encryption**). On average, attackers only need to make 256 SSL 3.0 requests to reveal one byte of encrypted messages.[76]

Although this vulnerability only exists in SSL 3.0 and most clients and servers support TLS 1.0 and above, all major browsers voluntarily downgrade to SSL 3.0 if the handshakes with newer versions of TLS fail unless they provide the option for a user or administrator to disable SSL 3.0

and the user or administrator does so. Therefore, the man-in-the-middle can first conduct a version rollback attack and then exploit this vulnerability.[76]

On December 8, 2014, a variant of POODLE was announced that impacts TLS implementations that do not properly enforce padding byte requirements.[112]

RC4 attacks

Despite the existence of attacks on RC4 that broke its security, cipher suites in SSL and TLS that were based on RC4 were still considered secure prior to 2013 based on the way in which they were used in SSL and TLS. In 2011, the RC4 suite was actually recommended as a work around for the BEAST attack.[113] New forms of attack disclosed in March 2013 conclusively demonstrated the feasibility of breaking RC4 in TLS, suggesting it was not a good workaround for BEAST.[75] An attack scenario was proposed by AlFardan, Bernstein, Paterson, Poettering and Schuldt that used newly discovered statistical biases in the RC4 key table[114] to recover parts of the plaintext with a large number of TLS encryptions.[115][116] An attack on RC4 in TLS and SSL that requires 13×2^{20} encryptions to break RC4 was unveiled on 8 July 2013 and later described as "feasible" in the accompanying presentation at a USENIX Security Symposium in August 2013.[117][118] In July 2015, subsequent improvements in the attack make it increasingly practical to defeat the security of RC4-encrypted TLS.[119]

As many modern browsers have been designed to defeat BEAST attacks (except Safari for Mac OS X 10.7 or earlier, for iOS 6 or earlier, and for Windows; see § Web browsers), RC4 is no longer a good choice for TLS 1.0. The CBC ciphers which were affected by the BEAST attack in the past have become a more popular choice for protection.[69] Mozilla and Microsoft recommend disabling RC4 where possible.[120][121] RFC 7465 (https://datatracker.ietf.org/doc/html/rfc7465) prohibits the use of RC4 cipher suites in all versions of TLS.

On September 1, 2015, Microsoft, Google and Mozilla announced that RC4 cipher suites would be disabled by default in their browsers (Microsoft Edge, Internet Explorer 11 on Windows 7/8.1/10, Firefox, and Chrome) in early 2016.[122][123][124]

Truncation attack

A TLS (logout) truncation attack blocks a victim's account logout requests so that the user unknowingly remains logged into a web service. When the request to sign out is sent, the attacker injects an unencrypted TCP FIN message (no more data from sender) to close the connection. The server therefore doesn't receive the logout request and is unaware of the abnormal termination.[125]

Published in July 2013,[126][127] the attack causes web services such as Gmail and Hotmail to display a page that informs the user that they have successfully signed-out, while ensuring that the user's browser maintains authorization with the service, allowing an attacker with subsequent access to the browser to access and take over control of the user's logged-in account. The attack does not rely on installing malware on the victim's computer; attackers need only place themselves between the victim and the web server (e.g., by setting up a rogue wireless hotspot).[125] This vulnerability also requires access to the victim's computer. Another possibility is when using FTP the data connection can have a false FIN in the data stream, and if the protocol rules for exchanging close_notify alerts is not adhered to a file can be truncated.

Unholy PAC attack

This attack, discovered in mid-2016, exploits weaknesses in the Web Proxy Autodiscovery Protocol (WPAD) to expose the URL that a web user is attempting to reach via a TLS-enabled web link.[128] Disclosure of a URL can violate a user's privacy, not only because of the website accessed, but also because URLs are sometimes used to authenticate users. Document sharing services, such as those offered by Google and Dropbox, also work by sending a user a security token that's included in the URL. An attacker who obtains such URLs may be able to gain full access to a victim's account or data.

The exploit works against almost all browsers and operating systems.

Sweet32 attack

The Sweet32 attack breaks all 64-bit block ciphers used in CBC mode as used in TLS by exploiting a birthday attack and either a man-in-the-middle attack or injection of a malicious JavaScript into a web page. The purpose of the man-in-the-middle attack or the JavaScript injection is to allow the attacker to capture enough traffic to mount a birthday attack.[129]

Implementation errors: Heartbleed bug, BERserk attack, Cloudflare bug

The Heartbleed bug is a serious vulnerability specific to the implementation of SSL/TLS in the popular OpenSSL cryptographic software library, affecting versions 1.0.1 to 1.0.1f. This weakness, reported in April 2014, allows attackers to steal private keys from servers that should normally be protected.[130] The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret private keys associated with the public certificates used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.[131] The vulnerability is caused by a buffer over-read bug in the OpenSSL software, rather than a defect in the SSL or TLS protocol specification.

In September 2014, a variant of Daniel Bleichenbacher's PKCS#1 v1.5 RSA Signature Forgery vulnerability[132] was announced by Intel Security Advanced Threat Research. This attack, dubbed BERserk, is a result of incomplete ASN.1 length decoding of public key signatures in some SSL implementations, and allows a man-in-the-middle attack by forging a public key signature.[133]

In February 2015, after media reported the hidden pre-installation of superfish adware on some Lenovo notebooks,[134] a researcher found a trusted root certificate on affected Lenovo machines to be insecure, as the keys could easily be accessed using the company name, Komodia, as a passphrase.[135] The Komodia library was designed to intercept client-side TLS/SSL traffic for parental control and surveillance, but it was also used in numerous adware programs, including Superfish, that were often surreptitiously installed unbeknownst to the computer user. In turn, these potentially unwanted programs installed the corrupt root certificate, allowing attackers to completely control web traffic and confirm false websites as authentic.

In May 2016, it was reported that dozens of Danish HTTPS-protected websites belonging to Visa Inc. were vulnerable to attacks allowing hackers to inject malicious code and forged content into the browsers of visitors.[136] The attacks worked because the TLS implementation used on the affected servers incorrectly reused random numbers (nonces) that are intended to be used only once, ensuring that each TLS handshake is unique.[136]

In February 2017, an implementation error caused by a single mistyped character in code used to parse HTML created a buffer overflow error on Cloudflare servers. Similar in its effects to the Heartbleed bug discovered in 2014, this overflow error, widely known as Cloudbleed, allowed unauthorized third parties to read data in the memory of programs running on the servers—data that should otherwise have been protected by TLS.[137]

Survey of websites vulnerable to attacks

As of July 2021, the Trustworthy Internet Movement estimated the ratio of websites that are vulnerable to TLS attacks.[74]

Survey of the TLS vulnerabilities of the most popular websites

Attacks	Security			
	Insecure	Depends	Secure	Other
Renegotiation attack	0.1% support insecure renegotiation	<0.1% support both	99.2% support secure renegotiation	0.7% no support
RC4 attacks	0.4% support RC4 suites used with modern browsers	6.5% support some RC4 suites	93.1% no support	—
TLS Compression (CRIME attack)	>0.0% vulnerable	—	—	—
Heartbleed	>0.0% vulnerable	—	—	—
ChangeCipherSpec injection attack	0.1% vulnerable and exploitable	0.2% vulnerable, not exploitable	98.5% not vulnerable	1.2% unknown
POODLE attack against TLS **(Original POODLE against SSL 3.0 is not included)**	0.1% vulnerable and exploitable	0.1% vulnerable, not exploitable	99.8% not vulnerable	0.2% unknown
Protocol downgrade	6.6% Downgrade defence not supported	—	72.3% Downgrade defence supported	21.0% unknown

Forward secrecy

Forward secrecy is a property of cryptographic systems which ensures that a session key derived from a set of public and private keys will not be compromised if one of the private keys is compromised in the future.[138] Without forward secrecy, if the server's private key is compromised, not only will all future TLS-encrypted sessions using that server certificate be compromised, but also any past sessions that used it as well (provided of course that these past sessions were intercepted and stored at the time of transmission).[139] An implementation of

TLS can provide forward secrecy by requiring the use of ephemeral Diffie–Hellman key exchange to establish session keys, and some notable TLS implementations do so exclusively: e.g., Gmail and other Google HTTPS services that use OpenSSL.[140] However, many clients and servers supporting TLS (including browsers and web servers) are not configured to implement such restrictions.[141][142] In practice, unless a web service uses Diffie–Hellman key exchange to implement forward secrecy, all of the encrypted web traffic to and from that service can be decrypted by a third party if it obtains the server's master (private) key; e.g., by means of a court order.[143]

Even where Diffie–Hellman key exchange is implemented, server-side session management mechanisms can impact forward secrecy. The use of TLS session tickets (a TLS extension) causes the session to be protected by AES128-CBC-SHA256 regardless of any other negotiated TLS parameters, including forward secrecy ciphersuites, and the long-lived TLS session ticket keys defeat the attempt to implement forward secrecy.[144][145][146] Stanford University research in 2014 also found that of 473,802 TLS servers surveyed, 82.9% of the servers deploying ephemeral Diffie–Hellman (DHE) key exchange to support forward secrecy were using weak Diffie–Hellman parameters. These weak parameter choices could potentially compromise the effectiveness of the forward secrecy that the servers sought to provide.[147]

Since late 2011, Google has provided forward secrecy with TLS by default to users of its Gmail service, along with Google Docs and encrypted search, among other services.[148] Since November 2013, Twitter has provided forward secrecy with TLS to users of its service.[149] As of August 2019, about 80% of TLS-enabled websites are configured to use cipher suites that provide forward secrecy to most web browsers.[74]

TLS interception

TLS interception (or HTTPS interception if applied particularly to that protocol) is the practice of intercepting an encrypted data stream in order to decrypt it, read and possibly manipulate it, and then re-encrypt it and send the data on its way again. This is done by way of a "transparent proxy": the interception software terminates the incoming TLS connection, inspects the HTTP plaintext, and then creates a new TLS connection to the destination.[150]

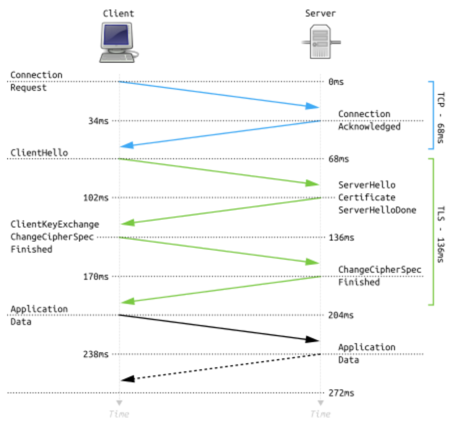
TLS/HTTPS interception is used as an information security measure by network operators in order to be able to scan for and protect against the intrusion of malicious content into the network, such as computer viruses and other malware.[150] Such content could otherwise not be detected as long as it is protected by encryption, which is increasingly the case as a result of the routine use of HTTPS and other secure protocols.

A significant drawback of TLS/HTTPS interception is that it introduces new security risks of its own. One notable limitation is that it provides a point where network traffic is available unencrypted thus giving attackers an incentive to attack this point in particular in order to gain access to otherwise secure content. The interception also allows the network operator, or persons who gain access to its interception system, to perform man-in-the-middle attacks against network users. A 2017 study found that "HTTPS interception has become startlingly widespread, and that interception products as a class have a dramatically negative impact on connection security".[150]

Protocol details

The TLS protocol exchanges *records*, which encapsulate the data to be exchanged in a specific format (see below). Each record can be compressed, padded, appended with a message authentication code (MAC), or encrypted, all depending on the state of the connection. Each record has a *content type* field that designates the type of data encapsulated, a length field and a TLS version field. The data encapsulated may be control or procedural messages of the TLS itself, or simply the application data needed to be transferred by TLS. The specifications (cipher suite, keys etc.) required to exchange application data by TLS, are agreed upon in the "TLS handshake" between the client requesting the data and the server responding to requests. The protocol therefore defines both the structure of payloads transferred in TLS and the procedure to establish and monitor the transfer.

TLS handshake



Simplified illustration of the full TLS 1.2 handshake with timing information.

When the connection starts, the record encapsulates a "control" protocol – the handshake messaging protocol (*content type* 22). This protocol is used to exchange all the information required by both sides for the exchange of the actual application data by TLS. It defines the format of messages and the order of their exchange. These may vary according to the demands of the client and server – i.e., there are several possible procedures to set up the connection. This initial exchange results in a successful TLS connection (both parties ready to transfer application data with TLS) or an alert message (as specified below).

Basic TLS handshake

A typical connection example follows, illustrating a handshake where the server (but not the client) is authenticated by its certificate:

1. Negotiation phase:
    - A client sends a **ClientHello** message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and suggested compression methods. If the client is attempting to perform a resumed handshake, it may send a *session ID*. If the client can use Application-Layer Protocol Negotiation, it may include a list of supported application protocols, such as HTTP/2.
    - The server responds with a **ServerHello** message, containing the chosen protocol version, a random number, cipher suite and compression method from the choices offered by the client. To confirm or allow resumed handshakes the server may send a *session ID*. The chosen protocol version should be the highest that both the client and server support. For example, if the client supports TLS version 1.1 and the server

supports version 1.2, version 1.1 should be selected; version 1.2 should not be selected.

- The server sends its **Certificate** message (depending on the selected cipher suite, this may be omitted by the server).[151]
- The server sends its **ServerKeyExchange** message (depending on the selected cipher suite, this may be omitted by the server). This message is sent for all DHE, ECDHE and DH_anon cipher suites.[7]
- The server sends a **ServerHelloDone** message, indicating it is done with handshake negotiation.
- The client responds with a **ClientKeyExchange** message, which may contain a *PreMasterSecret*, public key, or nothing. (Again, this depends on the selected cipher.) This *PreMasterSecret* is encrypted using the public key of the server certificate.
- The client and server then use the random numbers and *PreMasterSecret* to compute a common secret, called the "master secret". All other key data (session keys such as IV, symmetric encryption key, MAC key[152]) for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.

2. The client now sends a **ChangeCipherSpec** record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption parameters were present in the server certificate). "The ChangeCipherSpec is itself a record-level protocol with content type of 20.
   - The client sends an authenticated and encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.
   - The server will attempt to decrypt the client's *Finished* message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
3. Finally, the server sends a **ChangeCipherSpec**, telling the client, "Everything I tell you from now on will be authenticated (and encrypted, if encryption was negotiated)."
   - The server sends its authenticated and encrypted **Finished** message.
   - The client performs the same decryption and verification procedure as the server did in the previous step.

4. Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be authenticated and optionally encrypted exactly like in their *Finished* message. Otherwise, the content type will return 25 and the client will not authenticate.

Client-authenticated TLS handshake

The following *full* example shows a client being authenticated (in addition to the server as in the example above; see mutual authentication) via TLS using certificates exchanged between both peers.

1. Negotiation Phase:
   - A client sends a **ClientHello** message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods.
   - The server responds with a **ServerHello** message, containing the chosen protocol version, a random number, cipher suite and compression method from the choices offered by the client. The server may also send a *session id* as part of the message to perform a resumed handshake.
   - The server sends its **Certificate** message (depending on the selected cipher suite, this may be omitted by the server).[151]
   - The server sends its **ServerKeyExchange** message (depending on the selected cipher suite, this may be omitted by the server). This message is sent for all DHE, ECDHE and DH_anon ciphersuites.[7]
   - The server sends a **CertificateRequest** message, to request a certificate from the client.
   - The server sends a **ServerHelloDone** message, indicating it is done with handshake negotiation.
   - The client responds with a **Certificate** message, which contains the client's certificate, but not its private key.
   - The client sends a **ClientKeyExchange** message, which may contain a *PreMasterSecret*, public key, or nothing. (Again, this depends on the selected cipher.) This *PreMasterSecret* is encrypted using the public key of the server certificate.

    - The client sends a **CertificateVerify** message, which is a signature over the previous handshake messages using the client's certificate's private key. This signature can be verified by using the client's certificate's public key. This lets the server know that the client has access to the private key of the certificate and thus owns the certificate.
    - The client and server then use the random numbers and *PreMasterSecret* to compute a common secret, called the "master secret". All other key data ("session keys") for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.
2. The client now sends a **ChangeCipherSpec** record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated). "The ChangeCipherSpec is itself a record-level protocol and has type 20 and not 22.
    - Finally, the client sends an encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.
    - The server will attempt to decrypt the client's *Finished* message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
3. Finally, the server sends a **ChangeCipherSpec**, telling the client, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated)."
    - The server sends its own encrypted **Finished** message.
    - The client performs the same decryption and verification procedure as the server did in the previous step.
4. Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be encrypted exactly like in their *Finished* message.

Resumed TLS handshake

Public key operations (e.g., RSA) are relatively expensive in terms of computational power. TLS provides a secure shortcut in the handshake mechanism to avoid these operations: resumed sessions. Resumed sessions are implemented using session IDs or session tickets.

Apart from the performance benefit, resumed sessions can also be used for single sign-on, as it guarantees that both the original session and any resumed session originate from the same client. This is of particular importance for the FTP over TLS/SSL protocol, which would otherwise suffer from a man-in-the-middle attack in which an attacker could intercept the contents of the secondary data connections.[153]

TLS 1.3 handshake

The TLS 1.3 handshake was condensed to only one round trip compared to the two round trips required in previous versions of TLS/SSL.

First the client sends a clientHello message to the server that contains a list of supported ciphers in order of the client's preference and makes a guess on what key algorithm will be used so that it can send a secret key to share if needed. By making a guess at what key algorithm will be used, the server eliminates a round trip. After receiving the clientHello, the server sends a serverHello with its key, a certificate, the chosen cipher suite and the finished message.

After the client receives the server's finished message, it now is coordinated with the server on which cipher suite to use.[154]

Session IDs

In an ordinary *full* handshake, the server sends a *session id* as part of the **ServerHello** message. The client associates this *session id* with the server's IP address and TCP port, so that when the client connects again to that server, it can use the *session id* to shortcut the handshake. In the server, the *session id* maps to the cryptographic parameters previously negotiated, specifically the "master secret". Both sides must have the same "master secret" or the resumed handshake will fail (this prevents an eavesdropper from using a *session id*). The random data in the **ClientHello** and **ServerHello** messages virtually guarantee that the generated connection keys will be different from in the previous connection. In the RFCs, this type of handshake is called an *abbreviated* handshake. It is also described in the literature as a *restart* handshake.

1. Negotiation phase:
    - A client sends a **ClientHello** message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods. Included in the message is the *session id* from the previous TLS connection.
    - The server responds with a **ServerHello** message, containing the chosen protocol version, a random number, cipher suite and compression method from the choices offered by the client. If the server recognizes the *session id* sent by the client, it responds with the same *session id*. The client uses this to recognize that a resumed handshake is being performed. If the server does not recognize the *session id* sent by

the client, it sends a different value for its *session id*. This tells the client that a resumed handshake will not be performed. At this point, both the client and server have the "master secret" and random data to generate the key data to be used for this connection.

2. The server now sends a **ChangeCipherSpec** record, essentially telling the client, "Everything I tell you from now on will be encrypted." The ChangeCipherSpec is itself a record-level protocol and has type 20 and not 22.
   - Finally, the server sends an encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.
   - The client will attempt to decrypt the server's *Finished* message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
3. Finally, the client sends a **ChangeCipherSpec**, telling the server, "Everything I tell you from now on will be encrypted."
   - The client sends its own encrypted **Finished** message.
   - The server performs the same decryption and verification procedure as the client did in the previous step.
4. Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be encrypted exactly like in their *Finished* message.

Session tickets

RFC 5077 (https://datatracker.ietf.org/doc/html/rfc5077) extends TLS via use of session tickets, instead of session IDs. It defines a way to resume a TLS session without requiring that session-specific state is stored at the TLS server.

When using session tickets, the TLS server stores its session-specific state in a session ticket and sends the session ticket to the TLS client for storing. The client resumes a TLS session by sending the session ticket to the server, and the server resumes the TLS session according to the session-specific state in the ticket. The session ticket is encrypted and authenticated by the server, and the server verifies its validity before using its contents.

One particular weakness of this method with OpenSSL is that it always limits encryption and authentication security of the transmitted TLS session ticket to `AES128-CBC-SHA256`, no matter what other TLS parameters were negotiated for the actual TLS session.[145] This means that the state information (the TLS session ticket) is not as well protected as the TLS session itself. Of particular concern is OpenSSL's storage of the keys in an application-wide context (`SSL_CTX`), i.e. for the life of the application, and not allowing for re-keying of the `AES128-CBC-SHA256` TLS session tickets without resetting the application-wide OpenSSL context (which is uncommon, error-prone and often requires manual administrative intervention).[146][144]

TLS record

This is the general format of all TLS records.

TLS record format, general

Offset	Byte+0	Byte+1	Byte+2	Byte+3
Byte 0	Content type	—		
Bytes 1–4	Legacy version		Length	
	(Major)	*(Minor)*	*(bits 15–8)*	*(bits 7–0)*
Bytes 5–(*m*−1)	Protocol message(s)			
Bytes *m*–(*p*−1)	MAC (optional)			
Bytes *p*–(*q*−1)	Padding (block ciphers only)			

Content type
This field identifies the Record Layer Protocol Type contained in this record.

Content types

Hex	Dec	Type
0×14	20	ChangeCipherSpec
0×15	21	Alert
0×16	22	Handshake
0×17	23	Application
0×18	24	Heartbeat

Legacy version
This field identifies the major and minor version of TLS prior to TLS 1.3 for the contained message. For a ClientHello message, this need not be the *highest* version supported by the client. For TLS 1.3 and later, this must to be set 0x0303 and application must send supported versions in an extra message extension block.

Versions

Major version	Minor version	Version type
3	0	SSL 3.0
3	1	TLS 1.0
3	2	TLS 1.1
3	3	TLS 1.2
3	4	TLS 1.3

Length
The length of "protocol message(s)", "MAC" and "padding" fields combined (i.e. $q-5$), not to exceed 2^{14} bytes (16 KiB).
Protocol message(s)
One or more messages identified by the Protocol field. Note that this field may be encrypted depending on the state of the connection.
MAC and padding
A message authentication code computed over the "protocol message(s)" field, with additional key material included. Note that this field may be encrypted, or not included entirely, depending on the state of the connection.
No "MAC" or "padding" fields can be present at end of TLS records before all cipher algorithms and parameters have been negotiated and handshaked and then confirmed by sending a CipherStateChange record (see below) for signalling that these parameters will take effect in all further records sent by the same peer.

Handshake protocol

Most messages exchanged during the setup of the TLS session are based on this record, unless an error or warning occurs and needs to be signaled by an Alert protocol record (see below), or the encryption mode of the session is modified by another record (see ChangeCipherSpec protocol below).

TLS record format for handshake protocol

Offset	Byte+0	Byte+1	Byte+2	Byte+3
Byte 0	22	—		
Bytes 1–4	Legacy version		Length	
	(Major)	*(Minor)*	*(bits 15–8)*	*(bits 7–0)*
Bytes 5–8	Message type	Handshake message data length		
		(bits 23–16)	*(bits 15–8)*	*(bits 7–0)*
Bytes 9–(*n*–1)	Handshake message data			
Bytes *n*–(*n*+3)	Message type	Handshake message data length		
		(bits 23–16)	*(bits 15–8)*	*(bits 7–0)*
Bytes (*n*+4)–	Handshake message data			

Message type

This field identifies the handshake message type.

Message types

Code	Description
0	HelloRequest
1	ClientHello
2	ServerHello
4	NewSessionTicket
8	EncryptedExtensions (TLS 1.3 only)
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	Finished

Handshake message data length

This is a 3-byte field indicating the length of the handshake data, not including the header.

Note that multiple handshake messages may be combined within one record.

Alert protocol

This record should normally not be sent during normal handshaking or application exchanges. However, this message can be sent at any time during the handshake and up to the closure of the session. If this is used to signal a fatal error, the session will be closed immediately after sending this record, so this record is used to give a reason for this closure. If the alert level is flagged as a warning, the remote can decide to close the session if it decides that the session is not reliable enough for its needs (before doing so, the remote may also send its own signal).

TLS record format for alert protocol

Offset	Byte+0	Byte+1	Byte+2	Byte+3
Byte 0	21	—		
Bytes 1–4	Legacy version		Length	
	(Major)	*(Minor)*	0	2
Bytes 5–6	Level	Description	—	
Bytes 7–(*p*–1)	MAC (optional)			
Bytes *p*–(*q*–1)	Padding (block ciphers only)			

Level

This field identifies the level of alert. If the level is fatal, the sender should close the session immediately. Otherwise, the recipient may decide to terminate the session itself, by sending its own fatal alert and closing the session itself immediately after sending it. The use of Alert records is optional, however if it is missing before the session closure, the session may be resumed automatically (with its handshakes).

Normal closure of a session after termination of the transported application should preferably be alerted with at least the *Close notify* Alert type (with a simple warning level) to prevent such automatic resume of a new session. Signalling explicitly the normal closure of a secure session before effectively closing its transport layer is useful to prevent or detect attacks (like attempts to truncate the securely transported data, if it intrinsically does not have a predetermined length or duration that the recipient of the secured data may expect).

Alert level types

Code	Level type	Connection state
1	**warning**	connection or security may be unstable.
2	**fatal**	connection or security may be compromised, or an unrecoverable error has occurred.

Description

This field identifies which type of alert is being sent.

Alert description types

Code	Description	Level types	Note
0	Close notify	**warning/fatal**	
10	Unexpected message	**fatal**	
20	Bad record MAC	**fatal**	Possibly a bad SSL implementation, or payload has been tampered with e.g. FTP firewall rule on FTPS server.
21	Decryption failed	**fatal**	TLS only, reserved
22	Record overflow	**fatal**	TLS only
30	Decompression failure	**fatal**	
40	Handshake failure	**fatal**	
41	No certificate	**warning/fatal**	SSL 3.0 only, reserved
42	Bad certificate	**warning/fatal**	
43	Unsupported certificate	**warning/fatal**	e.g. certificate has only server authentication usage enabled and is presented as a client certificate
44	Certificate revoked	**warning/fatal**	
45	Certificate expired	**warning/fatal**	Check server certificate expire also check no certificate in the chain presented has expired
46	Certificate unknown	**warning/fatal**	
47	Illegal parameter	**fatal**	
48	Unknown CA (Certificate authority)	**fatal**	TLS only
49	Access denied	**fatal**	TLS only – e.g. no client certificate has been presented (TLS: Blank certificate message or SSLv3: No Certificate alert), but server is configured to require one.
50	Decode error	**fatal**	TLS only
51	Decrypt error	**warning/fatal**	TLS only

60	Export restriction	**fatal**	TLS only, reserved
70	Protocol version	**fatal**	TLS only
71	Insufficient security	**fatal**	TLS only
80	Internal error	**fatal**	TLS only
86	Inappropriate fallback	**fatal**	TLS only
90	User canceled	**fatal**	TLS only
100	No renegotiation	**warning**	TLS only
110	Unsupported extension	**warning**	TLS only
111	Certificate unobtainable	**warning**	TLS only
112	Unrecognized name	**warning/fatal**	TLS only; client's Server Name Indicator specified a hostname not supported by the server
113	Bad certificate status response	**fatal**	TLS only
114	Bad certificate hash value	**fatal**	TLS only
115	Unknown PSK identity (used in TLS-PSK and TLS-SRP)	**fatal**	TLS only
116	Certificate required	**fatal**	TLS version 1.3 only
120 or 255	No application protocol	**fatal**	TLS version 1.3 only

ChangeCipherSpec protocol

TLS record format for ChangeCipherSpec protocol

Offset	Byte+0	Byte+1	Byte+2	Byte+3
Byte 0	20	–		
Bytes 1–4	Legacy version		Length	
	(Major)	*(Minor)*	0	1
Byte 5	CCS protocol type	–		

CCS protocol type

Currently only 1.

Application protocol

TLS record format for application protocol

Offset	Byte+0	Byte+1	Byte+2	Byte+3
Byte 0	23	–		
Bytes 1–4	Legacy version		Length	
	(Major)	*(Minor)*	*(bits 15–8)*	*(bits 7–0)*
Bytes 5–(*m*−1)	Application data			
Bytes *m*–(*p*−1)	MAC (optional)			
Bytes *p*–(*q*−1)	Padding (block ciphers only)			

Length

Length of application data (excluding the protocol header and including the MAC and padding trailers)

MAC

32 bytes for the SHA-256-based HMAC, 20 bytes for the SHA-1-based HMAC, 16 bytes for the MD5-based HMAC.

Padding

Variable length; last byte contains the padding length.

Support for name-based virtual servers

From the application protocol point of view, TLS belongs to a lower layer, although the TCP/IP model is too coarse to show it. This means that the TLS handshake is usually (except in the STARTTLS case) performed before the application protocol can start. In the name-based virtual server feature being provided by the application layer, all co-hosted virtual servers share the same certificate because the server has to select and send a certificate immediately after the ClientHello message. This is a big problem in hosting environments because it means either sharing the same certificate among all customers or using a different IP address for each of them.

There are two known workarounds provided by X.509:

- If all virtual servers belong to the same domain, a wildcard certificate can be used.[155] Besides the loose host name selection that might be a problem or not, there is no common agreement about how to match wildcard certificates. Different rules are applied depending on the application protocol or software used.[156]
- Add every virtual host name in the subjectAltName extension. The major problem being that the certificate needs to be reissued whenever a new virtual server is added.

To provide the server name, RFC 4366 (https://datatracker.ietf.org/doc/html/rfc4366) Transport Layer Security (TLS) Extensions allow clients to include a Server Name Indication extension (SNI) in the extended ClientHello message. This extension hints to the server immediately which name the client wishes to connect to, so the server can select the appropriate certificate to send to the clients.

RFC 2817 (https://datatracker.ietf.org/doc/html/rfc2817) also documents a method to implement name-based virtual hosting by upgrading HTTP to TLS via an HTTP/1.1 Upgrade header. Normally this is to securely implement HTTP over TLS within the main "http" URI scheme (which avoids forking the URI space and reduces the number of used ports), however, few implementations currently support this.

Standards

Primary standards

The current approved version of TLS is version 1.3, which is specified in:

- RFC 8446 (https://datatracker.ietf.org/doc/html/rfc8446) : "The Transport Layer Security (TLS) Protocol Version 1.3".

The current standard replaces these former versions, which are now considered obsolete:

- RFC 2246 (https://datatracker.ietf.org/doc/html/rfc2246) : "The TLS Protocol Version 1.0".
- RFC 4346 (https://datatracker.ietf.org/doc/html/rfc4346) : "The Transport Layer Security (TLS) Protocol Version 1.1".
- RFC 5246 (https://datatracker.ietf.org/doc/html/rfc5246) : "The Transport Layer Security (TLS) Protocol Version 1.2".

As well as the never standardized SSL 2.0 and 3.0, which are considered obsolete:

- Internet Draft (1995) (https://tools.ietf.org/html/draft-hickman-netscape-ssl-00) , SSL Version 2.0
- RFC 6101 (https://datatracker.ietf.org/doc/html/rfc6101) : "The Secure Sockets Layer (SSL) Protocol Version 3.0".

Extensions

Other RFCs subsequently extended TLS.

Extensions to TLS 1.0 include:

- RFC 2595 (https://datatracker.ietf.org/doc/html/rfc2595) : "Using TLS with IMAP, POP3 and ACAP". Specifies an extension to the IMAP, POP3 and ACAP services that allow the server and client to use transport-layer security to provide private, authenticated communication over the Internet.
- RFC 2712 (https://datatracker.ietf.org/doc/html/rfc2712) : "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)". The 40-bit cipher suites defined in this memo appear only for the purpose of documenting the fact that those cipher suite codes have already been assigned.
- RFC 2817 (https://datatracker.ietf.org/doc/html/rfc2817) : "Upgrading to TLS Within HTTP/1.1", explains how to use the Upgrade mechanism in HTTP/1.1 to initiate Transport Layer Security (TLS) over an existing TCP connection. This allows unsecured and secured

HTTP traffic to share the same *well known* port (in this case, http: at 80 rather than https: at 443).

- RFC 2818 (https://datatracker.ietf.org/doc/html/rfc2818) : "HTTP Over TLS", distinguishes secured traffic from insecure traffic by the use of a different 'server port'.
- RFC 3207 (https://datatracker.ietf.org/doc/html/rfc3207) : "SMTP Service Extension for Secure SMTP over Transport Layer Security". Specifies an extension to the SMTP service that allows an SMTP server and client to use transport-layer security to provide private, authenticated communication over the Internet.
- RFC 3268 (https://datatracker.ietf.org/doc/html/rfc3268) : "AES Ciphersuites for TLS". Adds Advanced Encryption Standard (AES) cipher suites to the previously existing symmetric ciphers.
- RFC 3546 (https://datatracker.ietf.org/doc/html/rfc3546) : "Transport Layer Security (TLS) Extensions", adds a mechanism for negotiating protocol extensions during session initialisation and defines some extensions. Made obsolete by RFC 4366 (https://datatracker.ietf.org/doc/html/rfc4366) .
- RFC 3749 (https://datatracker.ietf.org/doc/html/rfc3749) : "Transport Layer Security Protocol Compression Methods", specifies the framework for compression methods and the DEFLATE compression method.
- RFC 3943 (https://datatracker.ietf.org/doc/html/rfc3943) : "Transport Layer Security (TLS) Protocol Compression Using Lempel-Ziv-Stac (LZS)".
- RFC 4132 (https://datatracker.ietf.org/doc/html/rfc4132) : "Addition of Camellia Cipher Suites to Transport Layer Security (TLS)".
- RFC 4162 (https://datatracker.ietf.org/doc/html/rfc4162) : "Addition of SEED Cipher Suites to Transport Layer Security (TLS)".
- RFC 4217 (https://datatracker.ietf.org/doc/html/rfc4217) : "Securing FTP with TLS".
- RFC 4279 (https://datatracker.ietf.org/doc/html/rfc4279) : "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", adds three sets of new cipher suites for the TLS protocol to support authentication based on pre-shared keys.

Extensions to TLS 1.1 include:

- RFC 4347 (https://datatracker.ietf.org/doc/html/rfc4347) : "Datagram Transport Layer Security" specifies a TLS variant that works over datagram protocols (such as UDP).

- RFC 4366 (https://datatracker.ietf.org/doc/html/rfc4366) : "Transport Layer Security (TLS) Extensions" describes both a set of specific extensions and a generic extension mechanism.
- RFC 4492 (https://datatracker.ietf.org/doc/html/rfc4492) : "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)".
- RFC 4680 (https://datatracker.ietf.org/doc/html/rfc4680) : "TLS Handshake Message for Supplemental Data".
- RFC 4681 (https://datatracker.ietf.org/doc/html/rfc4681) : "TLS User Mapping Extension".
- RFC 4785 (https://datatracker.ietf.org/doc/html/rfc4785) : "Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS)".
- RFC 5054 (https://datatracker.ietf.org/doc/html/rfc5054) : "Using the Secure Remote Password (SRP) Protocol for TLS Authentication". Defines the TLS-SRP ciphersuites.
- RFC 5077 (https://datatracker.ietf.org/doc/html/rfc5077) : "Transport Layer Security (TLS) Session Resumption without Server-Side State".
- RFC 5081 (https://datatracker.ietf.org/doc/html/rfc5081) : "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication", obsoleted by RFC 6091 (https://datatracker.ietf.org/doc/html/rfc6091) .

Extensions to TLS 1.2 include:

- RFC 5288 (https://datatracker.ietf.org/doc/html/rfc5288) : "AES Galois Counter Mode (GCM) Cipher Suites for TLS".
- RFC 5289 (https://datatracker.ietf.org/doc/html/rfc5289) : "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)".
- RFC 5746 (https://datatracker.ietf.org/doc/html/rfc5746) : "Transport Layer Security (TLS) Renegotiation Indication Extension".
- RFC 5878 (https://datatracker.ietf.org/doc/html/rfc5878) : "Transport Layer Security (TLS) Authorization Extensions".
- RFC 5932 (https://datatracker.ietf.org/doc/html/rfc5932) : "Camellia Cipher Suites for TLS"
- RFC 6066 (https://datatracker.ietf.org/doc/html/rfc6066) : "Transport Layer Security (TLS) Extensions: Extension Definitions", includes Server Name Indication and OCSP stapling.
- RFC 6091 (https://datatracker.ietf.org/doc/html/rfc6091) : "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication".

- RFC 6176 (https://datatracker.ietf.org/doc/html/rfc6176) : "Prohibiting Secure Sockets Layer (SSL) Version 2.0".
- RFC 6209 (https://datatracker.ietf.org/doc/html/rfc6209) : "Addition of the ARIA Cipher Suites to Transport Layer Security (TLS)".
- RFC 6347 (https://datatracker.ietf.org/doc/html/rfc6347) : "Datagram Transport Layer Security Version 1.2".
- RFC 6367 (https://datatracker.ietf.org/doc/html/rfc6367) : "Addition of the Camellia Cipher Suites to Transport Layer Security (TLS)".
- RFC 6460 (https://datatracker.ietf.org/doc/html/rfc6460) : "Suite B Profile for Transport Layer Security (TLS)".
- RFC 6655 (https://datatracker.ietf.org/doc/html/rfc6655) : "AES-CCM Cipher Suites for Transport Layer Security (TLS)".
- RFC 7027 (https://datatracker.ietf.org/doc/html/rfc7027) : "Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS)".
- RFC 7251 (https://datatracker.ietf.org/doc/html/rfc7251) : "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS".
- RFC 7301 (https://datatracker.ietf.org/doc/html/rfc7301) : "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension".
- RFC 7366 (https://datatracker.ietf.org/doc/html/rfc7366) : "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)".
- RFC 7465 (https://datatracker.ietf.org/doc/html/rfc7465) : "Prohibiting RC4 Cipher Suites".
- RFC 7507 (https://datatracker.ietf.org/doc/html/rfc7507) : "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks".
- RFC 7568 (https://datatracker.ietf.org/doc/html/rfc7568) : "Deprecating Secure Sockets Layer Version 3.0".
- RFC 7627 (https://datatracker.ietf.org/doc/html/rfc7627) : "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension".
- RFC 7685 (https://datatracker.ietf.org/doc/html/rfc7685) : "A Transport Layer Security (TLS) ClientHello Padding Extension".

Encapsulations of TLS include:

- RFC 5216 (https://datatracker.ietf.org/doc/html/rfc5216) : "The EAP-TLS Authentication Protocol"

Informational RFCs

- RFC 7457 (https://datatracker.ietf.org/doc/html/rfc7457) : "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)"
- RFC 7525 (https://datatracker.ietf.org/doc/html/rfc7525) : "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)"

See also

- Application-Layer Protocol Negotiation – a TLS extension used for SPDY and TLS False Start
- Bullrun (decryption program) – a secret anti-encryption program run by the U.S. National Security Agency
- Certificate authority
- Certificate Transparency
- HTTP Strict Transport Security – HSTS
- Key ring file
- Private Communications Technology (PCT) – a historic Microsoft competitor to SSL 2.0
- QUIC (Quick UDP Internet Connections) – "…was designed to provide security protection equivalent to TLS/SSL"; QUIC's main goal is to improve perceived performance of connection-oriented web applications that are currently using TCP
- Server-Gated Cryptography
- tcpcrypt
- DTLS
- TLS acceleration

References

1. *Lawrence, Scott; Khare, Rohit (May 2000). "Upgrading to TLS Within HTTP/1.1" (https://tools.ietf.org/html/rfc2817) . Internet Engineering Task Force. Retrieved December 15, 2018.*

2. "SSL/TLS in Detail" (https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc785811(v=ws.10)) . TechNet. Microsoft Docs. Retrieved October 24, 2021.
3. Hooper, Howard (2012). CCNP Security VPN 642-648 Official Cert Guide (https://books.google.com/books?id=5PJisOKJ0k8C&pg=PA22) (2 ed.). Cisco Press. p. 22. ISBN 9780132966382.
4. Spott, Andrew; Leek, Tom; et al. "What layer is TLS?" (https://security.stackexchange.com/a/93338) . Information Security Stack Exchange.
5. T. Dierks, E. Rescorla (August 2008). "Introduction" (https://datatracker.ietf.org/doc/html/rfc5246#section-1) . The Transport Layer Security (TLS) Protocol Version 1.2 (https://datatracker.ietf.org/doc/html/rfc5246) . sec. 1. doi:10.17487/RFC5246 (https://doi.org/10.17487%2FRFC5246) . RFC 5246 (https://datatracker.ietf.org/doc/html/rfc5246) .
6. E. Rescorla (August 2008). "The Transport Layer Security (TLS) Protocol Version 1.3" (https://tools.ietf.org/html/rfc8446) .
7. T. Dierks; E. Rescorla (August 2008). "The Transport Layer Security (TLS) Protocol Version 1.2" (http://tools.ietf.org/html/rfc5246) . Archived (https://web.archive.org/web/20171224222709/https://tools.ietf.org/html/rfc5246) from the original on December 24, 2017.
8. Bright, Peter (October 17, 2018). "Apple, Google, Microsoft, and Mozilla come together to end TLS 1.0" (https://arstechnica.com/gadgets/2018/10/browser-vendors-unite-to-end-support-for-20-year-old-tls-1-0) . Retrieved October 17, 2018.
9. "Here is what is new and changed in Firefox 74.0 Stable - gHacks Tech News" (https://www.ghacks.net/2020/03/10/here-is-what-is-new-and-changed-in-firefox-74-0-stable) . www.ghacks.net. March 10, 2020. Retrieved March 10, 2020.
10. "TLS 1.0 and TLS 1.1 - Chrome Platform Status" (https://chromestatus.com/feature/5759116003770368) . chromestatus.com. Retrieved March 10, 2020.
11. Deirks, Tim (August 2008). "The Transport Layer Security (TLS) Protocol Version 1.2" (https://www.ietf.org/rfc/rfc5246.txt) . IETF. Archived (https://web.archive.org/web/20080915032215/www.ietf.org/rfc/rfc5246.txt) from the original on September 15, 2008. Retrieved August 24, 2022.
12. "Using TLS to protect data" (https://www.ncsc.gov.uk/guidance/using-tls-to-protect-data) . www.ncsc.gov.uk. Archived (https://web.archive.org/web/20210721072543/ncsc.gov.uk/guidance/using-tls-to-protect-data) from the original on July 21, 2021. Retrieved August 24, 2022.
13. "TLS 1.3: One Year Later" (https://www.ietf.org/blog/tls13-adoption) . IETF. Archived (https://web.archive.org/web/20200708030455/https://www.ietf.org/blog/tls13-adoption) from the original on July 8, 2020. Retrieved August 24, 2022.
14. "Creating TLS: The Pioneering Role of Ruth Nelson" (https://www.circleid.com/posts/20190124_creating_tls_the_pioneering_role_of_ruth_nelson) .

15. Thomas Y. C. Woo, Raghuram Bindignavle, Shaowen Su and Simon S. Lam, SNP: An interface for secure network programming (https://www.cs.utexas.edu/users/lam/Vita/Cpapers/WBSL94.pdf) Proceedings USENIX Summer Technical Conference, June **1994**

16. Messmer, Ellen. "Father of SSL, Dr. Taher Elgamal, Finds Fast-Moving IT Projects in the Middle East" (https://web.archive.org/web/20140531105537/http://www.networkworld.com/news/2012/120412-elgamal-264739.html) . Network World. Archived from the original (http://www.networkworld.com/news/2012/120412-elgamal-264739.html) on May 31, 2014. Retrieved May 30, 2014.

17. Greene, Tim. "Father of SSL says despite attacks, the security linchpin has lots of life left" (https://web.archive.org/web/20140531105257/http://www.networkworld.com/news/2011/101111-elgamal-251806.html) . Network World. Archived from the original (http://www.networkworld.com/news/2011/101111-elgamal-251806.html) on May 31, 2014. Retrieved May 30, 2014.

18. Oppliger, Rolf (2016). "Introduction" (https://books.google.com/books?id=jm6uDgAAQBAJ&pg=PA15) . SSL and TLS: Theory and Practice (2nd ed.). Artech House. p. 13. ISBN 978-1-60807-999-5. Retrieved March 1, 2018 – via Google Books.

19. "THE SSL PROTOCOL" (https://web.archive.org/web/19970614020952/http://home.netscape.com/newsref/std/SSL.html) . Netscape Corporation. 2007. Archived from the original (http://home.netscape.com/newsref/std/SSL.html) on June 14, 1997.

20. Rescorla 2001

21. "POODLE: SSLv3 vulnerability (CVE-2014-3566)" (https://access.redhat.com/articles/1232123) . Archived (https://web.archive.org/web/20141205124712/https://access.redhat.com/articles/1232123) from the original on December 5, 2014. Retrieved October 21, 2014.

22. "Security Standards and Name Changes in the Browser Wars" (http://tim.dierks.org/2014/05/security-standards-and-name-changes-in.html) . Retrieved February 29, 2020.

23. Laura K. Gray (December 18, 2015). "Date Change for Migrating from SSL and Early TLS" (https://blog.pcisecuritystandards.org/migrating-from-ssl-and-early-tls) . Payment Card Industry Security Standards Council blog. Retrieved April 5, 2018.

24. Company, Newtek - Your Business Solutions. "Changes to PCI Compliance are Coming June 30. Is Your Ecommerce Business Ready?" (https://www.forbes.com/sites/thesba/2018/05/30/changes-to-pci-compliance-are-coming-june-30-is-your-ecommerce-business-ready) . Forbes. Retrieved June 20, 2018.

25. Dierks, T. & E. Rescorla (April 2006). "The Transport Layer Security (TLS) Protocol Version 1.1" (http://tools.ietf.org/html/rfc5246#ref-TLS1.1) . RFC 4346 (https://tools.ietf.org/html/rfc4346) . Archived (https://web.archive.org/web/20171224222709/https://tools.ietf.org/html/rfc5246#ref-TLS1.1) from the original on December 24, 2017.

26. Polk, Tim; McKay, Kerry; Chokhani, Santosh (April 2014). "Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations" (https://web.archive.org/web/20140508025330/http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf) (PDF). National Institute of Standards and Technology. p. 67. Archived from the original (http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf) (PDF) on May 8, 2014. Retrieved May 7, 2014.

27. "Twitter will deprecate support for TLS 1.0, TLS 1.1 on July 15" (https://www.thesslstore.com/blog/twitter-will-deprecate-support-for-tls-1-0-tls-1-1-on-july-15) . Hashed Out by The SSL Store. July 12, 2019. Retrieved June 14, 2021.

28. Mackie, Kurt. "Microsoft Delays End of Support for TLS 1.0 and 1.1 -" (https://mcpmag.com/articles/2020/04/02/microsoft-tls-1-0-and-1-1.aspx) . Microsoft Certified Professional Magazine Online.

29. "TLS 1.2 FAQ – Knowledge Base" (https://answers.psionline.com/knowledgebase/tls-1-2-faq) . Answers.psionline.com. Retrieved February 20, 2022.

30. T. Dierks, E. Rescorla (August 2008). "Finished" (https://datatracker.ietf.org/doc/html/rfc5246#section-7.4.9) . The Transport Layer Security (TLS) Protocol Version 1.2 (https://datatracker.ietf.org/doc/html/rfc5246) . sec. 7.4.9. doi:10.17487/RFC5246 (https://doi.org/10.17487%2FRFC5246) . RFC 5246 (https://datatracker.ietf.org/doc/html/rfc5246) .

31. "Differences between TLS 1.2 and TLS 1.3 (#TLS13)" (https://web.archive.org/web/20190919000200/https://www.wolfssl.com/differences-between-tls-12-and-tls-13-9) . WolfSSL. September 18, 2019. Archived from the original (https://www.wolfssl.com/differences-between-tls-12-and-tls-13-9) on September 19, 2019. Retrieved September 18, 2019.

32. "NSS 3.29 release notes" (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/NSS_3.29_release_notes) . Mozilla Developer Network. February 2017. Archived (https://web.archive.org/web/20170222052829/https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/NSS_3.29_release_notes) from the original on February 22, 2017.

33. "Enable TLS 1.3 by default" (https://bugzilla.mozilla.org/show_bug.cgi?id=1310516) . Bugzilla@Mozilla. October 16, 2016. Retrieved October 10, 2017.

34. "Firefox — Notes (60.0)" (https://www.mozilla.org/en-US/firefox/60.0/releasenotes) . Mozilla. Retrieved May 10, 2018.

35. "ProxySG, ASG and WSS will interrupt SSL connections when clients using TLS 1.3 access sites also using TLS 1.3" (https://bluecoat.force.com/knowledgebase/articles/Technical_Alert/000032878) . BlueTouch Online. May 16, 2017. Archived (https://web.archive.org/web/20170912061432/http://bluecoat.force.com/knowledgebase/articles/Technical_Alert/000032878) from the original on September 12, 2017. Retrieved September 11, 2017.

36. "TLS 1.3 IETF 100 Hackathon" (https://web.archive.org/web/20180115220635/https://datatracker.ietf.org/meeting/100/materials/slides-100-hackathon-sessa-tls-13) . Archived from the original (https://datatracker.ietf.org/meeting/100/materials/slides-100-hackathon-sessa-tls-13) on January 15, 2018.

37. IETF – Internet Engineering Task Force (November 12, 2017), IETF Hackathon Presentations and Awards (https://ghostarchive.org/varchive/youtube/20211028/33XW5yzjtME) , archived from the original (https://www.youtube.com/watch?v=33XW5yzjtME&t=2338) on October 28, 2021, retrieved November 14, 2017

38. "Hurrah! TLS 1.3 is here. Now to implement it and put it into software" (https://www.theregister.co.uk/2018/03/27/with_tls_13_signed_off_its_implementation_time) . Retrieved March 28, 2018.

39. IETF - Internet Engineering Task Force (July 15, 2018), IETF102-HACKATHON-20180715-1400 (https://ghostarchive.org/varchive/youtube/20211028/u6rz4PWA_As) , archived from the original (https://www.youtube.com/watch?v=u6rz4PWA_As&t=4526) on October 28, 2021, retrieved July 18, 2018

40. "wolfSSL TLS 1.3 BETA Release Now Available" (https://www.wolfssl.com/wolfssl-tls-1-3-beta-release-now-available) . info@wolfssl.com. May 11, 2017. Retrieved May 11, 2017.

41. "TLS 1.3 PROTOCOL SUPPORT" (https://www.wolfssl.com/docs/tls13) . info@wolfssl.com.

42. "TLS 1.3 Draft 28 Support in wolfSSL" (https://www.wolfssl.com/tls-1-3-draft-28-support-wolfssl) . info@wolfssl.com. June 14, 2018. Retrieved June 14, 2018.

43. "OpenSSL 1.1.1 Is Released" (https://www.openssl.org/blog/blog/2018/09/11/release111) . Matt Caswell. September 11, 2018. Retrieved December 19, 2018.

44. "Protocols in TLS/SSL (Schannel SSP)" (https://docs.microsoft.com/en-us/windows/win32/secauthn/protocols-in-tls-ssl--schannel-ssp) . Microsoft Docs. Retrieved November 24, 2021.

45. Hoffman-Andrews, Jacob (February 26, 2019). "ETS Isn't TLS and You Shouldn't Use It" (https://www.eff.org/deeplinks/2019/02/ets-isnt-tls-and-you-shouldnt-use-it) . Electronic Frontier Foundation. Retrieved February 27, 2019.

46. TS 103 523-3 - V1.1.1 - CYBER; Middlebox Security Protocol; Part 3: Profile for enterprise network and data centre access control (https://www.etsi.org/deliver/etsi_ts/103500_103599/10352303/01.01.01_60/ts_10352303v010101p.pdf#page=5) (PDF). ETSI.org. Archived (https://web.archive.org/web/20181114104718/https://www.etsi.org/deliver/etsi_ts/103500_103599/10352303/01.01.01_60/ts_10352303v010101p.pdf) (PDF) from the original on November 14, 2018.

47. Cory Doctorow (February 26, 2019). "Monumental Recklessness" (https://boingboing.net/2019/02/26/monumental-recklessness.html) . Boing Boing. Archived (https://web.archive.org/web/20190227071044/boingboing.net/2019/02/26/monumental-recklessness.html) from the original on February 27, 2019.

48. Rea, Scott (2013). "Alternatives to Certification Authorities for a Secure Web" (https://www.rsaconference.com/writable/presentations/file_upload/sec-t02_final.pdf) (PDF). RSA Conference Asia Pacific. Archived (https://web.archive.org/web/20161007222635/https://www.rsaconference.com/writable/presentations/file_upload/sec-t02_final.pdf) (PDF) from the original on October 7, 2016. Retrieved September 7, 2016.

49. "Counting SSL certificates" (https://web.archive.org/web/20150516035536/http://news.netcraft.com/archives/2015/05/13/counting-ssl-certificates.html). Archived from the original (https://news.netcraft.com/archives/2015/05/13/counting-ssl-certificates.html) on May 16, 2015. Retrieved February 20, 2022.

50. Raymond, Art (August 3, 2017). "Lehi's DigiCert swallows web security competitor in $1 billion deal" (https://www.deseretnews.com/article/865686081/Lehis-DigiCert-swallows-web-security-competitor-in-1-billion-deal.html). Deseret News. Retrieved May 21, 2020.

51. "Market share trends for SSL certificate authorities" (https://w3techs.com/technologies/history_overview/ssl_certificate). W3Techs. Retrieved May 21, 2020.

52. Ryan Singel (March 24, 2010). "Law Enforcement Appliance Subverts SSL" (https://www.wired.com/threatlevel/2010/03/packet-forensics). wired.com. Archived (https://web.archive.org/web/20140412151324/http://www.wired.com/threatlevel/2010/03/packet-forensics) from the original on April 12, 2014.

53. Seth Schoen (March 24, 2010). "New Research Suggests That Governments May Fake SSL Certificates" (https://www.eff.org/deeplinks/2010/03/researchers-reveal-likelihood-governments-fake-ssl). EFF.org. Archived (https://web.archive.org/web/20100325223422/http://www.eff.org/deeplinks/2010/03/researchers-reveal-likelihood-governments-fake-ssl) from the original on March 25, 2010.

54. P. Eronen, Ed. (December 2005). "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)" (https://tools.ietf.org/html/rfc4279). Internet Engineering Task Force. RFC 4279 (https://tools.ietf.org/html/rfc4279). Archived (https://web.archive.org/web/20130905074722/http://tools.ietf.org/html/rfc4279) from the original on September 5, 2013. Retrieved September 9, 2013.

55. D. Taylor, Ed. (November 2007). "Using the Secure Remote Password (SRP) Protocol for TLS Authentication" (https://tools.ietf.org/html/rfc5054). Internet Engineering Task Force. RFC 5054 (https://tools.ietf.org/html/rfc5054). Archived (https://web.archive.org/web/20141207151603/http://tools.ietf.org/html/rfc5054) from the original on December 7, 2014. Retrieved December 21, 2014.

56. Gothard, Peter (July 31, 2013). "Google updates SSL certificates to 2048-bit encryption" (http://www.computing.co.uk/ctg/news/2285984/google-updates-ssl-certificates-to-2048bit-encryption). Computing. Incisive Media. Archived (https://web.archive.org/web/20130922082322/http://www.computing.co.uk/ctg/news/2285984/google-updates-ssl-certificates-to-2048bit-encryption) from the original on September 22, 2013. Retrieved September 9, 2013.

57. "The value of 2,048-bit encryption: Why encryption key length matters" (http://searchsecurity.techtarget.com/answer/From-1024-to-2048-bit-The-security-effect-of-encryption-key-length). SearchSecurity. Archived (https://web.archive.org/web/20180116081141/http://searchsecurity.techtarget.com/answer/From-1024-to-2048-bit-The-security-effect-of-encryption-key-length) from the original on January 16, 2018. Retrieved December 18, 2017.

58. Sean Turner (September 17, 2015). "Consensus: remove DSA from TLS 1.3" (https://www.ietf.org/mail-archive/web/tls/current/msg17680.html). Archived (https://web.archive.org/web/20151003193113/http://www.ietf.org/mail-archive/web/tls/current/msg17680.html) from the original on October 3, 2015.

59. *RFC 8422*

60. *"GOST 28147-89 Cipher Suites for Transport Layer Security (TLS)" (https://datatracker.ietf.org/doc/html/draft-chudov-cryptopro-cptls-04). IETF.org. December 8, 2008. Archived (https://web.archive.org/web/20081211144308/tools.ietf.org/html/draft-chudov-cryptopro-cptls-04) from the original on December 11, 2008.*

61. *RFC 5288 (https://datatracker.ietf.org/doc/html/rfc5288), 5289 (https://datatracker.ietf.org/doc/html/rfc5289)*

62. *RFC 6655 (https://datatracker.ietf.org/doc/html/rfc6655), 7251 (https://datatracker.ietf.org/doc/html/rfc7251)*

63. *RFC 6367 (https://datatracker.ietf.org/doc/html/rfc6367)*

64. *RFC 5932 (https://datatracker.ietf.org/doc/html/rfc5932), 6367 (https://datatracker.ietf.org/doc/html/rfc6367)*

65. *RFC 6209 (https://datatracker.ietf.org/doc/html/rfc6209)*

66. *RFC 4162 (https://datatracker.ietf.org/doc/html/rfc4162)*

67. *"On the Practical (In-)Security of 64-bit Block Ciphers — Collision Attacks on HTTP over TLS and OpenVPN" (https://sweet32.info/SWEET32_CCS16.pdf) (PDF). October 28, 2016. Archived (https://web.archive.org/web/20170424021101/https://sweet32.info/SWEET32_CCS16.pdf) (PDF) from the original on April 24, 2017. Retrieved June 8, 2017.*

68. *"NIST Special Publication 800-57 Recommendation for Key Management — Part 1: General (Revised)" (https://web.archive.org/web/20140606050814/http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf) (PDF). March 8, 2007. Archived from the original (http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf) (PDF) on June 6, 2014. Retrieved July 3, 2014.*

69. Qualys SSL Labs. *"SSL/TLS Deployment Best Practices" (https://www.ssllabs.com/projects/best-practices/index.html). Archived (https://web.archive.org/web/20150704101956/https://www.ssllabs.com/projects/best-practices/index.html) from the original on July 4, 2015. Retrieved June 2, 2015.*

70. *RFC 5469 (https://datatracker.ietf.org/doc/html/rfc5469)*

71. *RFC 7905 (https://datatracker.ietf.org/doc/html/rfc7905)*

72. *"RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2" (https://datatracker.ietf.org/doc/html/rfc5246). Datatracker.ietf.org. Retrieved February 20, 2022.*

73. *"Http vs https" (https://www.instantssl.com/ssl-certificate-products/https.html). Archived (https://web.archive.org/web/20150212105201/https://www.instantssl.com/ssl-certificate-products/https.html) from the original on February 12, 2015. Retrieved February 12, 2015.*

74. As of May 07, 2022. "SSL Pulse: Survey of the SSL Implementation of the Most Popular Websites" (https://www.ssllabs.com/ssl-pulse) . Qualys. Retrieved May 31, 2022.

75. ivanr (March 19, 2013). "RC4 in TLS is Broken: Now What?" (https://community.qualys.com/blogs/securitylabs/2013/03/19/rc4-in-tls-is-broken-now-what) . Qualsys Security Labs. Archived (https://web.archive.org/web/20130827044512/https://community.qualys.com/blogs/securitylabs/2013/03/19/rc4-in-tls-is-broken-now-what) from the original on August 27, 2013. Retrieved July 30, 2013.

76. Bodo Möller, Thai Duong & Krzysztof Kotowicz. "This POODLE Bites: Exploiting The SSL 3.0 Fallback" (https://www.openssl.org/~bodo/ssl-poodle.pdf) (PDF). Archived (https://web.archive.org/web/20141014224443/https://www.openssl.org/~bodo/ssl-poodle.pdf) (PDF) from the original on October 14, 2014. Retrieved October 15, 2014.

77. "Internet Explorer 11 has retired and is officially out of support—what you need to know" (https://blogs.windows.com/windowsexperience/2022/06/15/internet-explorer-11-has-retired-and-is-officially-out-of-support-what-you-need-to-know) . June 15, 2022. Retrieved June 15, 2022.

78. "Internet Explorer 11 desktop app support ended for certain versions of Windows 10" (https://docs.microsoft.com/lifecycle/announcements/internet-explorer-11-end-of-support-windows-10) . Retrieved June 17, 2022.

79. "Java Secure Socket Extension (JSSE) Reference Guide" (https://docs.oracle.com/en/java/javase/17/security/java-secure-socket-extension-jsse-reference-guide.html) . Oracle Help Center. Retrieved December 24, 2021.

80. Georgiev, Martin; Iyengar, Subodh; Jana, Suman; Anubhai, Rishita; Boneh, Dan; Shmatikov, Vitaly (2012). The most dangerous code in the world: validating SSL certificates in non-browser software. Proceedings of the 2012 ACM conference on Computer and communications security (http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf) (PDF). pp. 38–49. ISBN 978-1-4503-1651-4. Archived (https://web.archive.org/web/20171022194807/http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf) (PDF) from the original on October 22, 2017.

81. "The Use of the SIPS URI Scheme in the Session Initiation Protocol (SIP)" (https://tools.ietf.org/html/rfc5630) . RFC 5630 (https://tools.ietf.org/html/rfc5630) .

82. "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)" (https://tools.ietf.org/html/rfc7457) . RFC 7457 (https://tools.ietf.org/html/rfc7457) . Archived (https://web.archive.org/web/20160304201813/https://tools.ietf.org/html/rfc7457) from the original on March 4, 2016.

83. "CVE – CVE-2009-3555" (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555) . Archived (https://web.archive.org/web/20160104234608/http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555) from the original on January 4, 2016.

84. Rescorla, Eric (November 5, 2009). "Understanding the TLS Renegotiation Attack" (http://www.educatedguesswork.org/2009/11/understanding_the_tls_renegoti.html) . Educated Guesswork. Archived (https://web.archive.org/web/20120211120608/http://www.educatedguesswork.org/2009/11/understanding_the_tls_renegoti.html) from the original on February 11, 2012. Retrieved November 27, 2009.

85. "SSL_CTX_set_options SECURE_RENEGOTIATION" (https://www.openssl.org/docs/ssl/SSL_CTX_set_options.html#SECURE_RENEGOTIATION) . OpenSSL Docs. February 25, 2010. Archived (https://web.archive.org/web/20101126121933/http://openssl.org/docs/ssl/SSL_CTX_set_options.html#SECURE_RENEGOTIATION) from the original on November 26, 2010. Retrieved November 18, 2010.

86. "GnuTLS 2.10.0 released" (http://article.gmane.org/gmane.network.gnutls.general/2046) . GnuTLS release notes. June 25, 2010. Archived (https://web.archive.org/web/20151017033726/http://article.gmane.org/gmane.network.gnutls.general/2046) from the original on October 17, 2015. Retrieved July 24, 2011.

87. "NSS 3.12.6 release notes" (https://web.archive.org/web/20120306184633/https://developer.mozilla.org/NSS_3.12.6_release_notes) . NSS release notes. March 3, 2010. Archived from the original (https://developer.mozilla.org/NSS_3.12.6_release_notes) on March 6, 2012. Retrieved July 24, 2011.

88. A. Langley; N. Modadugu; B. Moeller (June 2, 2010). "Transport Layer Security (TLS) False Start" (http://tools.ietf.org/html/draft-bmoeller-tls-falsestart-00) . Internet Engineering Task Force. IETF. Archived (https://web.archive.org/web/20130905215608/http://tools.ietf.org/html/draft-bmoeller-tls-falsestart-00) from the original on September 5, 2013. Retrieved July 31, 2013.

89. Gruener, Wolfgang. "False Start: Google Proposes Faster Web, Chrome Supports It Already" (https://web.archive.org/web/20101007061707/http://www.conceivablytech.com/3299/products/false-start-google-proposes-faster-web-chrome-supports-it-already) . Archived from the original (http://www.conceivablytech.com/3299/products/false-start-google-proposes-faster-web-chrome-supports-it-already) on October 7, 2010. Retrieved March 9, 2011.

90. Smith, Brian. "Limited rollback attacks in False Start and Snap Start" (http://www.ietf.org/mail-archive/web/tls/current/msg06933.html) . Archived (https://web.archive.org/web/20110504014418/http://www.ietf.org/mail-archive/web/tls/current/msg06933.html) from the original on May 4, 2011. Retrieved March 9, 2011.

91. Dimcev, Adrian. "False Start" (http://www.carbonwind.net/blog/post/Random-SSLTLS-101-False-Start.aspx) . Random SSL/TLS 101. Archived (https://web.archive.org/web/20110504060256/http://www.carbonwind.net/blog/post/Random-SSLTLS-101-False-Start.aspx) from the original on May 4, 2011. Retrieved March 9, 2011.

92. Mavrogiannopoulos, Nikos; Vercautern, Frederik; Velichkov, Vesselin; Preneel, Bart (2012). A cross-protocol attack on the TLS protocol. Proceedings of the 2012 ACM conference on Computer and communications security (https://www.cosic.esat.kuleuven.be/publications/article-2216.pdf) (PDF). pp. 62–72. ISBN 978-1-4503-1651-4. Archived (https://web.archive.org/web/20150706104327/https://www.cosic.esat.kuleuven.be/publications/article-2216.pdf) (PDF) from the original on July 6, 2015.

93. "SMACK: State Machine AttaCKs" (https://www.smacktls.com) . Archived (https://web.archive.org/web/20150312074827/https://www.smacktls.com) from the original on March 12, 2015.

94. Goodin, Dan (May 20, 2015). "HTTPS-crippling attack threatens tens of thousands of Web and mail servers" (https://arstechnica.com/security/2015/05/https-crippling-attack-threatens-tens-of-thousands-of-web-and-mail-servers) . Ars Technica. Archived (https://web.archive.org/web/20170519130937/https://arstechnica.com/security/2015/05/https-crippling-attack-threatens-tens-of-thousands-of-web-and-mail-servers) from the original on May 19, 2017.

95. Leyden, John (March 1, 2016). "One-third of all HTTPS websites open to DROWN attack" (https://www.theregister.com/2016/03/01/drown_tls_protocol_flaw) . The Register. Archived (https://web.archive.org/web/20160301215536/http://www.theregister.co.uk/2016/03/01/drown_tls_protocol_flaw) from the original on March 1, 2016. Retrieved March 2, 2016.

96. "More than 11 million HTTPS websites imperiled by new decryption attack" (https://arstechnica.com/information-technology/2016/03/more-than-13-million-https-websites-imperiled-by-new-decryption-attack) . Ars Technica. March 2016. Archived (https://web.archive.org/web/20160301191108/http://arstechnica.com/security/2016/03/more-than-13-million-https-websites-imperiled-by-new-decryption-attack) from the original on March 1, 2016. Retrieved March 2, 2016.

97. Thai Duong & Juliano Rizzo (May 13, 2011). "Here Come The ⊕ Ninjas" (https://bug665814.bugzilla.mozilla.org/attachment.cgi?id=540839) . Archived (https://web.archive.org/web/20140603102506/https://bug665814.bugzilla.mozilla.org/attachment.cgi?id=540839) from the original on June 3, 2014.

98. Goodin, Dan (September 19, 2011). "Hackers break SSL encryption used by millions of sites" (https://www.theregister.co.uk/2011/09/19/beast_exploits_paypal_ssl) . The Register. Archived (https://web.archive.org/web/20120210185309/http://www.theregister.co.uk/2011/09/19/beast_exploits_paypal_ssl) from the original on February 10, 2012.

99. "Y Combinator comments on the issue" (http://news.ycombinator.com/item?id=3015498) . September 20, 2011. Archived (https://web.archive.org/web/20120331225714/http://news.ycombinator.com/item?id=3015498) from the original on March 31, 2012.

100. "Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures" (https://web.archive.org/web/20120630143111/http://www.openssl.org/~bodo/tls-cbc.txt) . May 20, 2004. Archived from the original (https://www.openssl.org/~bodo/tls-cbc.txt) on June 30, 2012.

101. Ristic, Ivan (September 10, 2013). "Is BEAST Still a Threat?" (https://community.qualys.com/blogs/securitylabs/2013/09/10/is-beast-still-a-threat). Archived (https://web.archive.org/web/20141012121824/https://community.qualys.com/blogs/securitylabs/2013/09/10/is-beast-still-a-threat) from the original on October 12, 2014. Retrieved October 8, 2014.

102. "Chrome Stable Release" (http://googlechromereleases.blogspot.jp/2011/10/chrome-stable-release.html). Chrome Releases. October 25, 2011. Archived (https://web.archive.org/web/20150220020306/http://googlechromereleases.blogspot.jp/2011/10/chrome-stable-release.html) from the original on February 20, 2015. Retrieved February 1, 2015.

103. "Attack against TLS-protected communications" (https://blog.mozilla.org/security/2011/09/27/attack-against-tls-protected-communications). Mozilla Security Blog. Mozilla. September 27, 2011. Archived (https://web.archive.org/web/20150304221307/https://blog.mozilla.org/security/2011/09/27/attack-against-tls-protected-communications) from the original on March 4, 2015. Retrieved February 1, 2015.

104. Smith, Brian (September 30, 2011). "(CVE-2011-3389) Rizzo/Duong chosen plaintext attack (BEAST) on SSL/TLS 1.0 (facilitated by websockets-76)" (https://bugzilla.mozilla.org/show_bug.cgi?id=665814).

105. MSRC (January 10, 2012). Vulnerability in SSL/TLS Could Allow Information Disclosure (2643584) (https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2012/ms12-006). Security Bulletins (Technical report). MS12-006. Retrieved October 24, 2021 – via Microsoft Docs.

106. Ristic, Ivan (October 31, 2013). "Apple Enabled BEAST Mitigations in OS X 10.9 Mavericks" (https://community.qualys.com/blogs/securitylabs/2013/10/31/apple-enabled-beast-mitigations-in-os-x-109-mavericks). Archived (https://web.archive.org/web/20141012122536/https://community.qualys.com/blogs/securitylabs/2013/10/31/apple-enabled-beast-mitigations-in-os-x-109-mavericks) from the original on October 12, 2014. Retrieved October 8, 2014.

107. Goodin, Dan (September 13, 2012). "Crack in Internet's foundation of trust allows HTTPS session hijacking" (https://arstechnica.com/security/2012/09/crime-hijacks-https-sessions). Ars Technica. Archived (https://web.archive.org/web/20130801104610/http://arstechnica.com/security/2012/09/crime-hijacks-https-sessions) from the original on August 1, 2013. Retrieved July 31, 2013.

108. Fisher, Dennis (September 13, 2012). "CRIME Attack Uses Compression Ratio of TLS Requests as Side Channel to Hijack Secure Sessions" (https://web.archive.org/web/20120915224635/http://threatpost.com/en_us/blogs/crime-attack-uses-compression-ratio-tls-requests-side-channel-hijack-secure-sessions-091312). ThreatPost. Archived from the original (http://threatpost.com/en_us/blogs/crime-attack-uses-compression-ratio-tls-requests-side-channel-hijack-secure-sessions-091312) on September 15, 2012. Retrieved September 13, 2012.

109. Goodin, Dan (August 1, 2013). "Gone in 30 seconds: New attack plucks secrets from HTTPS-protected pages" (https://arstechnica.com/security/2013/08/gone-in-30-seconds-new-attack-plucks-secrets-from-https-protected-pages). Ars Technica. Condé Nast. Archived (https://web.archive.org/web/20130803181144/http://arstechnica.com/security/2013/08/gone-in-30-seconds-new-attack-plucks-secrets-from-https-protected-pages) from the original on August 3, 2013. Retrieved August 2, 2013.

110. Leyden, John (August 2, 2013). "Step into the BREACH: New attack developed to read encrypted web data" (https://www.theregister.co.uk/2013/08/02/breach_crypto_attack). The Register. Archived (https://web.archive.org/web/20130805233414/http://www.theregister.co.uk/2013/08/02/breach_crypto_attack) from the original on August 5, 2013. Retrieved August 2, 2013.

111. P. Gutmann (September 2014). "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)" (http://tools.ietf.org/html/rfc7366). Internet Engineering Task Force. Archived (https://web.archive.org/web/20150512214100/https://tools.ietf.org/html/rfc7366) from the original on May 12, 2015.

112. Langley, Adam (December 8, 2014). "The POODLE bites again" (https://www.imperialviolet.org/2014/12/08/poodleagain.html). Archived (https://web.archive.org/web/20141208200653/https://www.imperialviolet.org/2014/12/08/poodleagain.html) from the original on December 8, 2014. Retrieved December 8, 2014.

113. "ssl - Safest ciphers to use with the BEAST? (TLS 1.0 exploit) I've read that RC4 is immune" (https://serverfault.com/questions/315042/safest-ciphers-to-use-with-the-beast-tls-1-0-exploit-ive-read-that-rc4-is-im). Serverfault.com. Retrieved February 20, 2022.

114. Pouyan Sepehrdad; Serge Vaudenay; Martin Vuagnoux (2011). "Discovery and Exploitation of New Biases in RC4". In Alex Biryukov; Guang Gong; Douglas R. Stinson (eds.). Selected Areas in Cryptography: 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers. Lecture Notes in Computer Science. Vol. 6544. pp. 74–91. doi:10.1007/978-3-642-19574-7_5 (https://doi.org/10.1007%2F978-3-642-19574-7_5). ISBN 978-3-642-19573-0.

115. Green, Matthew (March 12, 2013). "Attack of the week: RC4 is kind of broken in TLS" (http://blog.cryptographyengineering.com/2013/03/attack-of-week-rc4-is-kind-of-broken-in.html). Cryptography Engineering. Archived (https://web.archive.org/web/20130314214026/http://blog.cryptographyengineering.com/2013/03/attack-of-week-rc4-is-kind-of-broken-in.html) from the original on March 14, 2013. Retrieved March 12, 2013.

116. AlFardan, Nadhem; Bernstein, Dan; Paterson, Kenny; Poettering, Bertram; Schuldt, Jacob. "On the Security of RC4 in TLS" (http://www.isg.rhul.ac.uk/tls). Royal Holloway University of London. Archived (https://web.archive.org/web/20130315084623/http://www.isg.rhul.ac.uk/tls) from the original on March 15, 2013. Retrieved March 13, 2013.

117. AlFardan, Nadhem J.; Bernstein, Daniel J.; Paterson, Kenneth G.; Poettering, Bertram; Schuldt, Jacob C. N. (July 8, 2013). "On the Security of RC4 in TLS and WPA" (http://www.isg.rhul.ac.uk/tls/RC4biases.pdf) (PDF). Information Security Group. Archived (https://web.archive.org/web/20130922170155/http://www.isg.rhul.ac.uk/tls/RC4biases.pdf) (PDF) from the original on September 22, 2013. Retrieved September 2, 2013.

118. AlFardan, Nadhem J.; Bernstein, Daniel J.; Paterson, Kenneth G.; Poettering, Bertram; Schuldt, Jacob C. N. (August 15, 2013). On the Security of RC4 in TLS (https://www.usenix.org/sites/default/files/conference/protected-files/alfardan_sec13_slides.pdf) (PDF). 22nd USENIX Security Symposium. p. 51. Archived (https://web.archive.org/web/20130922133950/https://www.usenix.org/sites/default/files/conference/protected-files/alfardan_sec13_slides.pdf) (PDF) from the original on September 22, 2013. Retrieved September 2, 2013. "Plaintext recovery attacks against RC4 in TLS are feasible although not truly practical"

119. Goodin, Dan (July 15, 2015). "Once-theoretical crypto attack against HTTPS now verges on practicality" (https://arstechnica.com/security/2015/07/once-theoretical-crypto-attack-against-https-now-verges-on-practicality) . Ars Technical. Conde Nast. Archived (https://web.archive.org/web/20150716084138/http://arstechnica.com/security/2015/07/once-theoretical-crypto-attack-against-https-now-verges-on-practicality) from the original on July 16, 2015. Retrieved July 16, 2015.

120. "Mozilla Security Server Side TLS Recommended Configurations" (https://wiki.mozilla.org/Security/Server_Side_TLS) . Mozilla. Archived (https://web.archive.org/web/20150103093047/https://wiki.mozilla.org/Security/Server_Side_TLS) from the original on January 3, 2015. Retrieved January 3, 2015.

121. "Security Advisory 2868725: Recommendation to disable RC4" (http://blogs.technet.com/b/srd/archive/2013/11/12/security-advisory-2868725-recommendation-to-disable-rc4.aspx) . Microsoft. November 12, 2013. Archived (https://web.archive.org/web/20131118081816/http://blogs.technet.com/b/srd/archive/2013/11/12/security-advisory-2868725-recommendation-to-disable-rc4.aspx) from the original on November 18, 2013. Retrieved December 4, 2013.

122. "Ending support for the RC4 cipher in Microsoft Edge and Internet Explorer 11" (https://blogs.windows.com/msedgedev/2015/09/01/ending-support-for-the-rc4-cipher-in-microsoft-edge-and-internet-explorer-11) . Microsoft Edge Team. September 1, 2015. Archived (https://web.archive.org/web/20150902054341/http://blogs.windows.com/msedgedev/2015/09/01/ending-support-for-the-rc4-cipher-in-microsoft-edge-and-internet-explorer-11) from the original on September 2, 2015.

123. Langley, Adam (September 1, 2015). "Intent to deprecate: RC4" (https://groups.google.com/a/chromium.org/forum/#!msg/security-dev/kVfCywocUO8/vgi_rQuhKgAJ) .

124. Barnes, Richard (September 1, 2015). "Intent to ship: RC4 disabled by default in Firefox 44" (https://groups.google.com/forum/#!topic/mozilla.dev.platform/JIEFcrGhqSM/discussion) . Archived (http://arquivo.pt/wayback/20110122130054/https://groups.google.com/forum/#!topic/mozilla.dev.platform/JIEFcrGhqSM/discussion) from the original on January 22, 2011.

125. John Leyden (August 1, 2013). "Gmail, Outlook.com and e-voting 'pwned' on stage in crypto-dodge hack" (https://www.theregister.co.uk/2013/08/01/gmail_hotmail_hijacking) . The Register. Archived (https://web.archive.org/web/20130801193054/http://www.theregister.co.uk/2013/08/01/gmail_hotmail_hijacking) from the original on August 1, 2013. Retrieved August 1, 2013.

126. "BlackHat USA Briefings" (https://www.blackhat.com/us-13/briefings.html#Smyth) . Black Hat 2013. Archived (https://web.archive.org/web/20130730124037/http://www.blackhat.com/us-13/briefings.html#Smyth) from the original on July 30, 2013. Retrieved August 1, 2013.

127. Smyth, Ben; Pironti, Alfredo (2013). Truncating TLS Connections to Violate Beliefs in Web Applications (https://hal.inria.fr/hal-01102013) . 7th USENIX Workshop on Offensive Technologies (report). Archived (https://web.archive.org/web/20151106110117/https://hal.inria.fr/hal-01102013) from the original on November 6, 2015. Retrieved February 15, 2016.

128. Goodin, Dan (July 26, 2016). "New attack bypasses HTTPS protection on Macs, Windows, and Linux" (https://arstechnica.com/security/2016/07/new-attack-that-cripples-https-crypto-works-on-macs-windows-and-linux) . Ars Technica. Condé Nast. Archived (https://web.archive.org/web/20160727160434/http://arstechnica.com/security/2016/07/new-attack-that-cripples-https-crypto-works-on-macs-windows-and-linux) from the original on July 27, 2016. Retrieved July 28, 2016.

129. Goodin, Dan (August 24, 2016). "HTTPS and OpenVPN face new attack that can decrypt secret cookies" (https://arstechnica.com/security/2016/08/new-attack-can-pluck-secrets-from-1-of-https-traffic-affects-top-sites) . Ars Technica. Archived (https://web.archive.org/web/20160824181630/http://arstechnica.com/security/2016/08/new-attack-can-pluck-secrets-from-1-of-https-traffic-affects-top-sites) from the original on August 24, 2016. Retrieved August 24, 2016.

130. "Why is it called the 'Heartbleed Bug'?" (https://www.washingtonpost.com/blogs/style-blog/wp/2014/04/09/why-is-it-called-the-heartbleed-bug) . The Washington Post. April 9, 2014. Archived (https://web.archive.org/web/20141009063758/http://www.washingtonpost.com/blogs/style-blog/wp/2014/04/09/why-is-it-called-the-heartbleed-bug) from the original on October 9, 2014.

131. "Heartbleed Bug vulnerability [9 April 2014]" (https://blogs.comodo.com/e-commerce/heartbleed-bug-comodo-urges-openssl-users-to-apply-patch) . Comodo Group. Archived (https://web.archive.org/web/20140705212748/https://blogs.comodo.com/e-commerce/heartbleed-bug-comodo-urges-openssl-users-to-apply-patch) from the original on July 5, 2014.

132. Bleichenbacher, Daniel (August 2006). "Bleichenbacher's RSA signature forgery based on implementation error" (https://web.archive.org/web/20141216203704/http://www.imc.org/ietf-openpgp/mail-archive/msg06063.html) . Archived from the original (http://www.imc.org/ietf-openpgp/mail-archive/msg06063.html) on December 16, 2014.

133. "BERserk" (http://www.intelsecurity.com/advanced-threat-research) . Intel Security: Advanced Threat Research. September 2014. Archived (https://web.archive.org/web/20150112153121/http://www.intelsecurity.com/advanced-threat-research) from the original on January 12, 2015.

134. Goodin, Dan (February 19, 2015). "Lenovo PCs ship with man-in-the-middle adware that breaks HTTPS connections" (https://arstechnica.com/information-technology/2015/02/lenovo-pcs-ship-with-man-in-the-middle-adware-that-breaks-https-connections) . Ars Technica. Archived (https://web.archive.org/web/20170912103610/https://arstechnica.com/information-technology/2015/02/lenovo-pcs-ship-with-man-in-the-middle-adware-that-breaks-https-connections) from the original on September 12, 2017. Retrieved December 10, 2017.

135. Valsorda, Filippo (February 20, 2015). "Komodia/Superfish SSL validation is broken" (https://blog.filippo.io/komodia-superfish-ssl-validation-is-broken) . Filippo.io. Archived (https://web.archive.org/web/20150224112141/https://blog.filippo.io/komodia-superfish-ssl-validation-is-broken) from the original on February 24, 2015.

136. Goodin, Dan (May 26, 2016). " "Forbidden attack" makes dozens of HTTPS Visa sites vulnerable to tampering" (https://arstechnica.com/security/2016/05/faulty-https-settings-leave-dozens-of-visa-sites-vulnerable-to-forgery-attacks) . Ars Technica. Archived (https://web.archive.org/web/20160526175713/http://arstechnica.com/security/2016/05/faulty-https-settings-leave-dozens-of-visa-sites-vulnerable-to-forgery-attacks) from the original on May 26, 2016. Retrieved May 26, 2016.

137. Clark Estes, Adam (February 24, 2017). "Everything You Need to Know About Cloudbleed, the Latest Internet Security Disaster" (https://gizmodo.com/everything-you-need-to-know-about-cloudbleed-the-lates-1792710616) . Gizmodo. Archived (https://web.archive.org/web/20170225013516/http://gizmodo.com/everything-you-need-to-know-about-cloudbleed-the-lates-1792710616) from the original on February 25, 2017. Retrieved February 24, 2017.

138. Diffie, Whitfield; van Oorschot, Paul C; Wiener, Michael J. (June 1992). "Authentication and Authenticated Key Exchanges" (http://citeseer.ist.psu.edu/diffie92authentication.html) . Designs, Codes and Cryptography. **2** (2): 107–125. CiteSeerX 10.1.1.59.6682 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.6682) . doi:10.1007/BF00124891 (https://doi.org/10.1007%2FBF00124891) . S2CID 7356608 (https://api.semanticscholar.org/CorpusID:7356608) . Archived (https://web.archive.org/web/20080313081157/http://citeseer.ist.psu.edu/diffie92authentication.html) from the original on March 13, 2008. Retrieved February 11, 2008.

139. "Discussion on the TLS mailing list in October 2007" (https://web.archive.org/web/20130922103746/http://www.ietf.org/mail-archive/web/tls/current/msg02134.html) . Archived from the original (http://www1.ietf.org/mail-archive/web/tls/current/msg02134.html) on September 22, 2013. Retrieved February 20, 2022.

140. "Protecting data for the long term with forward secrecy" (http://googleonlinesecurity.blogspot.com.au/2011/11/protecting-data-for-long-term-with.html) . Archived (https://web.archive.org/web/20130506184654/http://googleonlinesecurity.blogspot.com.au/2011/11/protecting-data-for-long-term-with.html) from the original on May 6, 2013. Retrieved November 5, 2012.

141. Bernat, Vincent (November 28, 2011). "SSL/TLS & Perfect Forward Secrecy" (https://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html) . Archived (https://web.archive.org/web/20120827064047/http://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html) from the original on August 27, 2012. Retrieved November 5, 2012.

142. "SSL Labs: Deploying Forward Secrecy" (https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy) . Qualys.com. June 25, 2013. Archived (https://web.archive.org/web/20130626193314/https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy) from the original on June 26, 2013. Retrieved July 10, 2013.

143. Ristic, Ivan (August 5, 2013). "SSL Labs: Deploying Forward Secrecy" (https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy) . Qualsys. Archived (https://web.archive.org/web/20130920150259/https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy) from the original on September 20, 2013. Retrieved August 31, 2013.

144. Langley, Adam (June 27, 2013). "How to botch TLS forward secrecy" (https://www.imperialviolet.org/2013/06/27/botchingpfs.html) . imperialviolet.org. Archived (https://web.archive.org/web/20130808221614/https://www.imperialviolet.org/2013/06/27/botchingpfs.html) from the original on August 8, 2013.

145. Daignière, Florent. "TLS "Secrets": Whitepaper presenting the security implications of the deployment of session tickets (RFC 5077) as implemented in OpenSSL" (https://media.blackhat.com/us-13/US-13-Daigniere-TLS-Secrets-WP.pdf) (PDF). Matta Consulting Limited. Archived (https://web.archive.org/web/20130806233112/https://media.blackhat.com/us-13/US-13-Daigniere-TLS-Secrets-WP.pdf) (PDF) from the original on August 6, 2013. Retrieved August 7, 2013.

146. Daignière, Florent. "TLS "Secrets": What everyone forgot to tell you..." (https://media.blackhat.com/us-13/US-13-Daigniere-TLS-Secrets-Slides.pdf) (PDF). Matta Consulting Limited. Archived (https://web.archive.org/web/20130805134805/https://media.blackhat.com/us-13/US-13-Daigniere-TLS-Secrets-Slides.pdf) (PDF) from the original on August 5, 2013. Retrieved August 7, 2013.

147. L.S. Huang; S. Adhikarla; D. Boneh; C. Jackson (2014). "An Experimental Study of TLS Forward Secrecy Deployments" (http://crypto.stanford.edu/~dabo/pubs/abstracts/websec_ecc.html) . IEEE Internet Computing. **18** (6): 43–51. CiteSeerX 10.1.1.663.4653 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.663.4653) . doi:10.1109/MIC.2014.86 (https://doi.org/10.1109%2FMIC.2014.86) . S2CID 11264303 (https://api.semanticscholar.org/CorpusID:11264303) . Archived (https://web.archive.org/web/20150920011317/http://crypto.stanford.edu/~dabo/pubs/abstracts/websec_ecc.html) from the original on September 20, 2015. Retrieved October 16, 2015.

148. "Protecting data for the long term with forward secrecy" (http://googleonlinesecurity.blogspot.com.au/2011/11/protecting-data-for-long-term-with.html) . Archived (https://web.archive.org/web/20140212214518/http://googleonlinesecurity.blogspot.com.au/2011/11/protecting-data-for-long-term-with.html) from the original on February 12, 2014. Retrieved March 7, 2014.

149. Hoffman-Andrews, Jacob. "Forward Secrecy at Twitter" (https://blog.twitter.com/2013/forward-secrecy-at-twitter-0). Twitter. Archived (https://web.archive.org/web/20140216041202/https://blog.twitter.com/2013/forward-secrecy-at-twitter-0) from the original on February 16, 2014. Retrieved March 7, 2014.

150. Durumeric, Zakir; Ma, Zane; Springall, Drew; Barnes, Richard; Sullivan, Nick; Bursztein, Elie; Bailey, Michael; Halderman, J. Alex; Paxson, Vern (September 5, 2017). "The Security Impact of HTTPS Interception" (https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/security-impact-https-interception). NDSS Symposium. doi:10.14722/ndss.2017.23456 (https://doi.org/10.14722%2Fndss.2017.23456). ISBN 978-1-891562-46-4.

151. These certificates are currently X.509, but RFC 6091 (https://datatracker.ietf.org/doc/html/rfc6091) also specifies the use of OpenPGP-based certificates.

152. "tls - Differences between the terms "pre-master secret", "master secret", "private key", and "shared secret"?" (https://crypto.stackexchange.com/questions/27131/differences-between-the-terms-pre-master-secret-master-secret-private-key). Cryptography Stack Exchange. Retrieved October 1, 2020.

153. Chris (February 18, 2009). "vsftpd-2.1.0 released – Using TLS session resume for FTPS data connection authentication" (http://scarybeastsecurity.blogspot.com/2009/02/vsftpd-210-released.html). Scarybeastsecurity. blogspot.com. Archived (https://web.archive.org/web/20120707213409/http://scarybeastsecurity.blogspot.com/2009/02/vsftpd-210-released.html) from the original on July 7, 2012. Retrieved May 17, 2012.

154. Valsorda, Filippo (September 23, 2016). "An overview of TLS 1.3 and Q&A" (https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a). The Cloudflare Blog.

155. Wildcard SSL Certificate overview (https://ssl.comodo.com/wildcard-ssl-certificates.php), archived (https://web.archive.org/web/20150623231035/https://ssl.comodo.com/wildcard-ssl-certificates.php) from the original on June 23, 2015, retrieved July 2, 2015

156. Named-based SSL virtual hosts: how to tackle the problem (https://www.switch.ch/pki/meetings/2007-01/namebased_ssl_virtualhosts.pdf) (PDF), archived (https://web.archive.org/web/20120803022659/https://www.switch.ch/pki/meetings/2007-01/namebased_ssl_virtualhosts.pdf) (PDF) from the original on August 3, 2012, retrieved May 17, 2012

Further reading

Wikimedia Commons has media related to ***SSL and TLS***.

- Wagner, David; Schneier, Bruce (November 1996). "Analysis of the SSL 3.0 Protocol" (http://www.schneier.com/paper-ssl.pdf) (PDF). *The Second USENIX Workshop on Electronic Commerce Proceedings*. USENIX Press. pp. 29–40.

- Eric Rescorla (2001). *SSL and TLS: Designing and Building Secure Systems* (https://archive.org/details/ssltls00eric) . United States: Addison-Wesley Pub Co. ISBN 978-0-201-61598-2.
- Stephen A. Thomas (2000). *SSL and TLS essentials securing the Web*. New York: Wiley. ISBN 978-0-471-38354-3.
- Bard, Gregory (2006). "A Challenging But Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL" (http://eprint.iacr.org/2006/136) . *International Association for Cryptologic Research* (136). Retrieved September 23, 2011.
- Canvel, Brice. "Password Interception in a SSL/TLS Channel" (http://lasecwww.epfl.ch/memo/memo_ssl.shtml) . Retrieved April 20, 2007.
- IETF Multiple Authors. "RFC of change for TLS Renegotiation" (http://tools.ietf.org/html/rfc5746) . Retrieved December 11, 2009.
- Creating VPNs with IPsec and SSL/TLS (http://www.linuxjournal.com/article/9916) Linux Journal article by Rami Rosen
- Joshua Davies (2010). *Implementing SSL/TLS*. Wiley. ISBN 978-0470920411.
- Polk, Tim; McKay, Kerry; Chokhani, Santosh (April 2014). "Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations" (https://web.archive.org/web/20140508025330/http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf) (PDF). National Institute of Standards and Technology. Archived from the original (http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf) (PDF) on May 8, 2014. Retrieved May 7, 2014.
- Abdou, AbdelRahman; van Oorschot, Paul (August 2017). "Server Location Verification (SLV) and Server Location Pinning: Augmenting TLS Authentication" (https://dl.acm.org/citation.cfm?id=3139294) . *Transactions on Privacy and Security*. ACM. **21** (1): 1:1–1:26. doi:10.1145/3139294 (https://doi.org/10.1145%2F3139294) . S2CID 5869541 (https://api.semanticscholar.org/CorpusID:5869541) .
- Ivan Ristic (2022). *Bulletproof TLS and PKI, Second Edition*. Feisty Duck. ISBN 978-1907117091.

External links

- (Internet Engineering Task Force) TLS Workgroup (https://datatracker.ietf.org/wg/tls)