WSA - HW – 4 part 2 and 3:

1. Find and document information for the Transport Layer Security (TLS) cryptographic.

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols which are designed to provide communication security over the Internet. They use X.509 certificates and hence asymmetric cryptography to assure the counterparty with whom they are communicating, and to exchange a symmetric key. This session key is then used to encrypt data flowing between the parties. This allows for data/message confidentiality, and message authentication codes for message integrity and as a by-product message authentication.
Several versions of the protocols are in widespread use in applications such as web browsing, electronic mail, Internet faxing, instant messaging, and voice-over-IP (VoIP). An important property in this context is forward secrecy, so the short term session key cannot be derived from the long term asymmetric secret key.

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 beneficial 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.

In the TCP/IP model view, TLS and SSL encrypt the data of network connections at a lower sublayer of its application layer. In OSI model equivalences, TLS/SSL is initialized at layer 5 (the session layer) then works at layer 6 (the presentation layer): first the session layer has a handshake using an asymmetric cipher in order to establish cipher settings and a shared key for that session; then the presentation layer encrypts the rest of the communication using a symmetric cipher and that session key. In both models, TLS and SSL work on behalf of the underlying transport layer, whose segments carry encrypted data.

TLS is an IETF standards track protocol, first defined in 1999 and last updated in RFC 5246 (August 2008) and RFC 6176 (March 2011). It is based on the earlier SSL specifications (1994, 1995, 1996) developed by Netscape Communications for adding the HTTPS protocol to their Navigator web browser.

Description:

The TLS protocol allows client-server applications to communicate across a network in a way designed to prevent eavesdropping and tampering.

Since protocols can operate either with or without TLS (or SSL), it is necessary for the client to indicate to the server whether it wants to set up a TLS connection or not. There are two main ways of achieving this; one option is to use a different port number for TLS connections (for example port 443 for HTTPS). The other is to use the regular port number and have the client request that the server switch the connection to TLS using a protocol-specific mechanism (for example STARTTLS for mail and news protocols).

Once the client and server have decided to use TLS, they negotiate a stateful connection by using a handshaking procedure. During this handshake, the client and server agree on various parameters used to establish the connection's security:

1. The client sends the server the client's SSL version number, cipher settings, session-specific data, and other information that the server needs to communicate with the client using SSL.

2. The server sends the client the server's SSL version number, cipher settings, session-specific data, and other information that the client needs to communicate with the server over SSL. The server also sends its own certificate, and if the client is requesting a server resource that requires client authentication, the server requests the client's certificate.
3. The client uses the information sent by the server to authenticate the server e.g., in the case of a web browser connecting to a web server, the browser checks whether the received certificate's subject name actually matches the name of the server being contacted, whether the issuer of the certificate is a trusted certificate authority, whether the certificate has expired, and, ideally, whether the certificate has been revoked. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client proceeds to the next step.
4. Using all data generated in the handshake thus far, the client (with the cooperation of the server, depending on the cipher in use) creates the pre-master secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in step 2), and then sends the encrypted pre-master secret to the server.
5. If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case, the client sends both the signed data and the client's own certificate to the server along with the encrypted pre-master secret.
6. If the server has requested client authentication, the server attempts to authenticate the client. If the client cannot be authenticated, the session ends. If the client can be successfully authenticated, the server uses its private key to decrypt the pre-master secret, and then performs a series of steps (which the client also performs, starting from the same pre-master secret) to generate the master secret.
7. Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity (that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection).
8. The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
9. The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.

The SSL handshake is now complete and the session begins. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.

This is the normal operation condition of the secure channel. At any time, due to internal or external stimulus (either automation or user intervention), either side may renegotiate the connection, in which case, the process repeats itself.

This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the key material until the connection closes.

If any one of the above steps fails, the TLS handshake fails and the connection is not created.

In step 3, the client must check a chain of "signatures" from a "root of trust" built into, or added to, the client. The client must *also* check that none of these have been revoked; this is not often implemented correctly but is a requirement of any public-key authentication system. If the particular signer beginning this server's chain is trusted, and all signatures in the chain remain trusted, then the Certificate (thus the server) is trusted.

History and development:

TLS 1.0

TLS 1.0 was first defined in RFC 2246 in January 1999 as an upgrade of SSL Version 3.0. As stated in the RFC, "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant to preclude interoperability between TLS 1.0 and SSL 3.0. " TLS 1.0 does include a means by which a TLS implementation can downgrade the connection to SSL 3.0, thus weakening security.

TLS 1.1

TLS 1.1 was defined in RFC 4346 in April 2006. 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.

TLS 1.2

TLS 1.2 was defined in RFC 5246 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 is still truncated to 96-bits.
- The MD5-SHA-1 combination in the digitally signed element was replaced with a single hash negotiated during handshake, defaults to SHA-1.
- Enhancement in the client's and server's ability to specify which hash and signature algorithms they will accept.
- Expansion of support for authenticated encryption ciphers, used mainly for Galois/Counter Mode (GCM) and CCM mode of Advanced Encryption Standard encryption.
- TLS Extensions definition and Advanced Encryption Standard cipher suites were added.

All TLS versions were further refined in RFC 6176 in March 2011 removing their backward compatibility with SSL such that TLS sessions will never negotiate the use of Secure Sockets Layer (SSL) version 2.0.

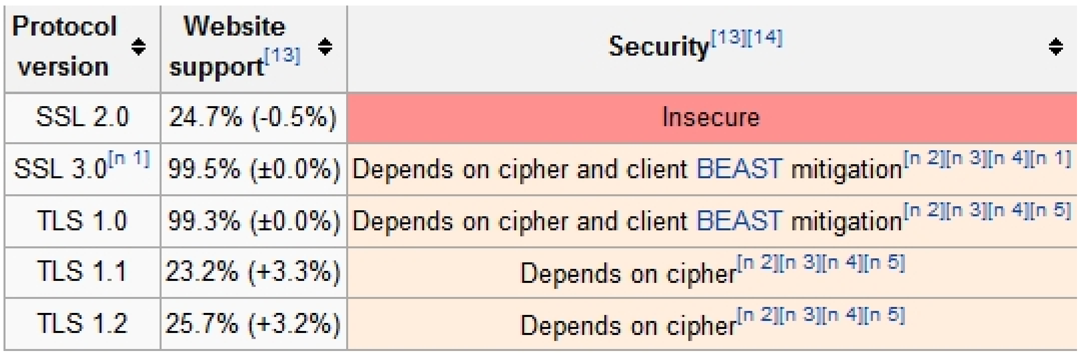
Applications and adoption:

In applications design, TLS is usually implemented on top of any of the Transport Layer protocols, encapsulating the application-specific protocols such as HTTP, FTP, SMTP, NNTP and XMPP. Historically it 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 which has been standardized independently using the term Datagram Transport Layer Security (DTLS).

Websites:

A prominent use of TLS is for securing World Wide Web traffic between the website and the browser carried by HTTP to form HTTPS. Notable applications are electronic commerce and asset management.

Website protocol support



Protocol version	Website support[13]	Security[13][14]
SSL 2.0	24.7% (-0.5%)	Insecure
SSL 3.0[n 1]	99.5% (±0.0%)	Depends on cipher and client BEAST mitigation[n 2][n 3][n 4][n 1]
TLS 1.0	99.3% (±0.0%)	Depends on cipher and client BEAST mitigation[n 2][n 3][n 4][n 5]
TLS 1.1	23.2% (+3.3%)	Depends on cipher[n 2][n 3][n 4][n 5]
TLS 1.2	25.7% (+3.2%)	Depends on cipher[n 2][n 3][n 4][n 5]

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. Among the methods used for key exchange/agreement are:

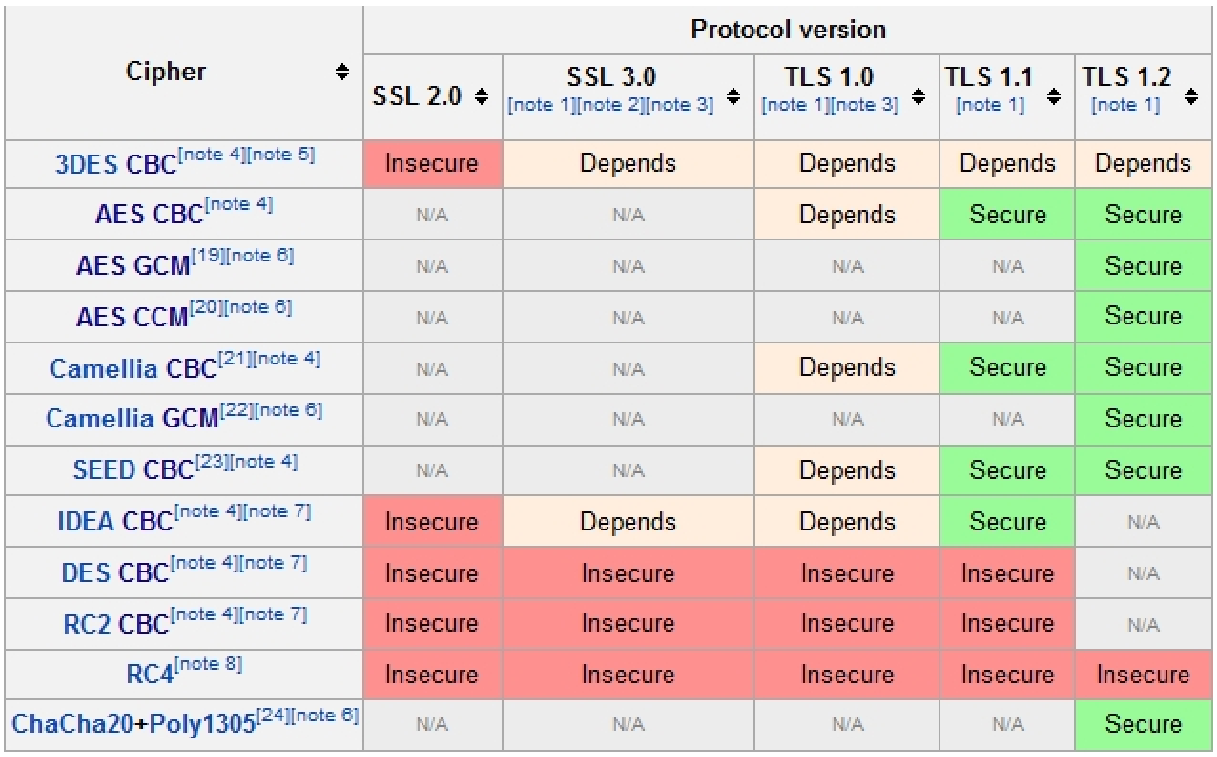
- public and private keys generated with RSA (denoted TLS_RSA in the TLS handshake protocol)
- Diffie-Hellman (denoted TLS_DH in the TLS handshake protocol)
- ephemeral Diffie-Hellman (denoted TLS_DHE in the handshake protocol)
- Elliptic Curve Diffie-Hellman (denoted TLS_ECDH), ephemeral Elliptic Curve Diffie-Hellman (TLS_ECDHE)
- anonymous Diffie-Hellman (TLS_DH_anon)
- PSK (TLS_PSK)

The TLS_DH_anon key agreement method does not authenticate the server or the user and hence is rarely used. 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.

Cipher security against publicly known feasible attacks:

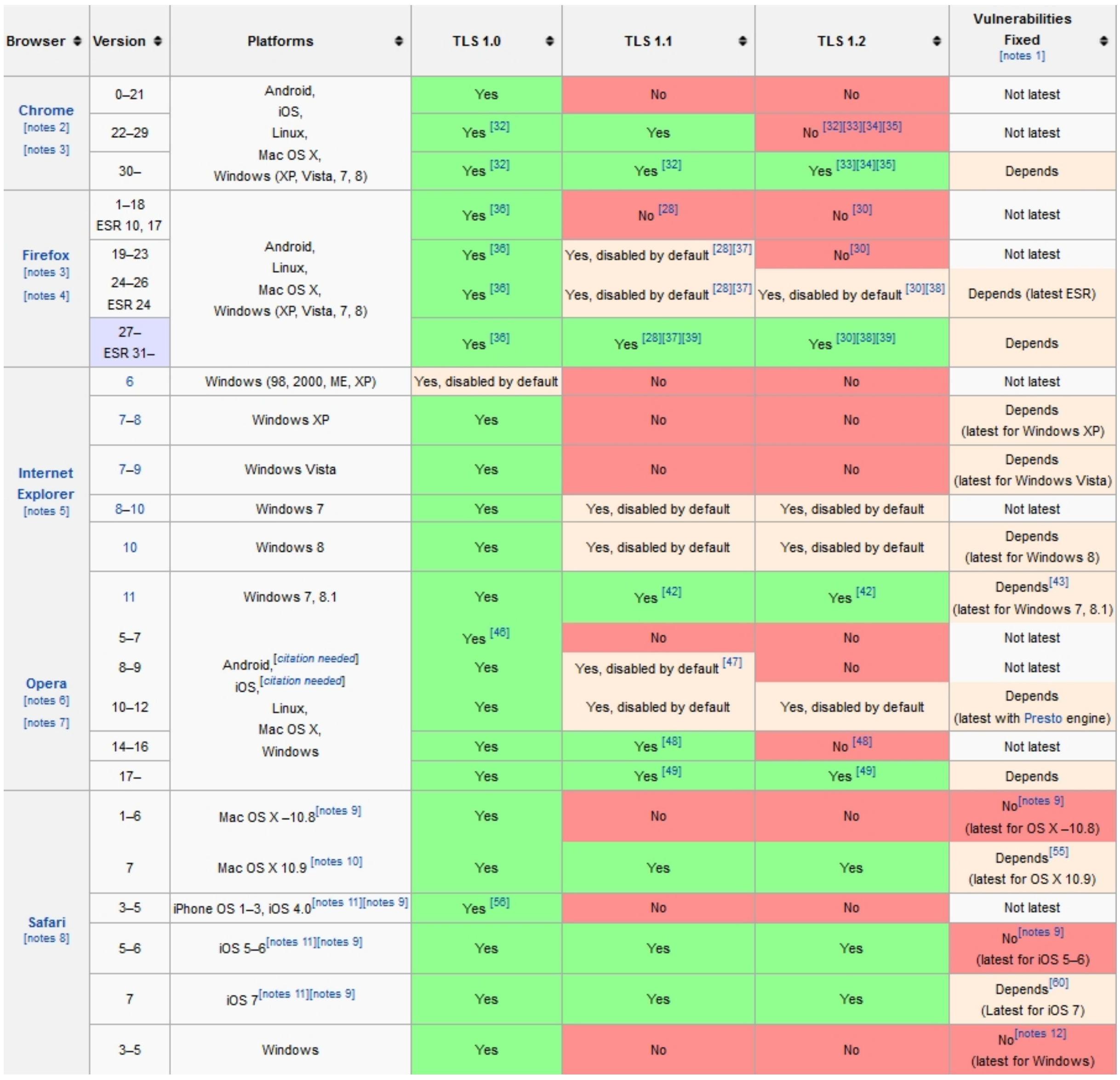


Cipher	Protocol version				
	SSL 2.0	SSL 3.0 [note 1][note 2][note 3]	TLS 1.0 [note 1][note 3]	TLS 1.1 [note 1]	TLS 1.2 [note 1]
3DES CBC[note 4][note 5]	Insecure	Depends	Depends	Depends	Depends
AES CBC[note 4]	N/A	N/A	Depends	Secure	Secure
AES GCM[19][note 6]	N/A	N/A	N/A	N/A	Secure
AES CCM[20][note 6]	N/A	N/A	N/A	N/A	Secure
Camellia CBC[21][note 4]	N/A	N/A	Depends	Secure	Secure
Camellia GCM[22][note 6]	N/A	N/A	N/A	N/A	Secure
SEED CBC[23][note 4]	N/A	N/A	Depends	Secure	Secure
IDEA CBC[note 4][note 7]	Insecure	Depends	Depends	Secure	N/A
DES CBC[note 4][note 7]	Insecure	Insecure	Insecure	Insecure	N/A
RC2 CBC[note 4][note 7]	Insecure	Insecure	Insecure	Insecure	N/A
RC4[note 8]	Insecure	Insecure	Insecure	Insecure	Insecure
ChaCha20+Poly1305[24][note 6]	N/A	N/A	N/A	N/A	Secure

Web browsers:

As of January 2014, the latest version of all major web browsers support SSL 3.0, TLS 1.0, 1.1, and 1.2. However, Mozilla Firefox still disables TLS 1.1 and 1.2 by default, and Internet Explorer for Windows Vista or older and Safari for Mac OS X 10.8 or older support only SSL 3.0 and TLS 1.0.

Browser support for TLS



Browser	Version	Platforms	TLS 1.0	TLS 1.1	TLS 1.2	Vulnerabilities Fixed [notes 1]
Chrome [notes 2] [notes 3]	0–21	Android, iOS, Linux, Mac OS X, Windows (XP, Vista, 7, 8)	Yes	No	No	Not latest
	22–29		Yes [32]	Yes	No [32][33][34][35]	Not latest
	30–		Yes [32]	Yes [32]	Yes [33][34][35]	Depends
Firefox [notes 3] [notes 4]	1–18 ESR 10, 17	Android, Linux, Mac OS X, Windows (XP, Vista, 7, 8)	Yes [36]	No [28]	No [30]	Not latest
	19–23		Yes [36]	Yes, disabled by default [28][37]	No[30]	Not latest
	24–26 ESR 24		Yes [36]	Yes, disabled by default [28][37]	Yes, disabled by default [30][38]	Depends (latest ESR)
	27– ESR 31–		Yes [36]	Yes [28][37][39]	Yes [30][38][39]	Depends
Internet Explorer [notes 5]	6	Windows (98, 2000, ME, XP)	Yes, disabled by default	No	No	Not latest
	7–8	Windows XP	Yes	No	No	Depends (latest for Windows XP)
	7–9	Windows Vista	Yes	No	No	Depends (latest for Windows Vista)
	8–10	Windows 7	Yes	Yes, disabled by default	Yes, disabled by default	Not latest
	10	Windows 8	Yes	Yes, disabled by default	Yes, disabled by default	Depends (latest for Windows 8)
	11	Windows 7, 8.1	Yes	Yes [42]	Yes [42]	Depends[43] (latest for Windows 7, 8.1)
Opera [notes 6] [notes 7]	5–7	Android,[citation needed] iOS,[citation needed] Linux, Mac OS X, Windows	Yes [46]	No	No	Not latest
	8–9		Yes	Yes, disabled by default [47]	No	Not latest
	10–12		Yes	Yes, disabled by default	Yes, disabled by default	Depends (latest with Presto engine)
	14–16		Yes	Yes [48]	No [48]	Not latest
	17–		Yes	Yes [49]	Yes [49]	Depends
Safari [notes 8]	1–6	Mac OS X –10.8[notes 9]	Yes	No	No	No[notes 9] (latest for OS X –10.8)
	7	Mac OS X 10.9 [notes 10]	Yes	Yes	Yes	Depends[55] (latest for OS X 10.9)
	3–5	iPhone OS 1–3, iOS 4.0[notes 11][notes 9]	Yes [56]	No	No	Not latest
	5–6	iOS 5–6[notes 11][notes 9]	Yes	Yes	Yes	No[notes 9] (latest for iOS 5–6)
	7	iOS 7[notes 11][notes 9]	Yes	Yes	Yes	Depends[60] (Latest for iOS 7)
	3–5	Windows	Yes	No	No	No[notes 12] (latest for Windows)

TLS has a variety of security measures:

- Protection against a downgrade of the protocol to a previous (less secure) version or a weaker cipher suite.
- Numbering subsequent Application records with a sequence number and using this sequence number in the message authentication codes (MACs).
- Using a message digest enhanced with a key (so only a key-holder can check the MAC). The HMAC construction used by most TLS cipher suites is specified in RFC 2104 (SSL 3.0 used a different hash-based MAC).
- The message that ends the handshake ("Finished") sends a hash of all the exchanged handshake messages seen by both parties.

- The pseudorandom function splits the input data in half and processes each one with a different hashing algorithm (MD5 and SHA-1), then XORs them together to create the MAC. This provides protection even if one of these algorithms is found to be vulnerable.

Attacks against TLS:

- **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. 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. This extension has become a proposed standard and has been assigned the number RFC 5746. The RFC has been implemented by several libraries.

- **Version rollback attacks**

Modifications to the original protocols, like **False Start** (adopted and enabled by Google Chrome) or Snap Start, have been reported to introduce limited TLS protocol version rollback attacks or to allow modifications to the cipher suite list sent by the client to the server (an attacker may be able to influence the cipher suite selection in an attempt to downgrade the cipher suite strength, to use either a weaker symmetric encryption algorithm or a weaker key exchange). It has been shown in the Association for Computing Machinery (ACM) conference on computer and communications security that the False Start extension is at risk as in certain circumstances it could allow an attacker to recover the encryption keys offline and access the encrypted data.

- **BEAST attack**

On September 23, 2011 researchers Thai Duong and Juliano Rizzo demonstrated a proof of concept called **BEAST** (**Browser Exploit Against SSL/TLS**) using a Java applet to violate same origin policy constraints, for a long-known cipher block chaining (CBC) vulnerability in TLS 1.0. Practical exploits had not been previously demonstrated for this vulnerability, which was originally discovered by Phillip Rogaway 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.

Mozilla updated the development versions of 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.

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.

Users of Windows 7, Windows 8 and Windows Server 2008 R2 can enable use of TLS 1.1 and 1.2, but this workaround will fail if it is not supported by the other end of the connection and will result in a fall-back to TLS 1.0.

- **CRIME and BREACH attacks**

CRIME (security exploit) and BREACH (security exploit)

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. 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 wide-spread than SPDY and TLS compression combined. In 2013 a new instance of the CRIME attack against HTTP compression, dubbed BREACH, was announced. Built 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). All versions of TLS and SSL are at risk from BREACH regardless of the encryption algorithm or cipher used. 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. 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.

- **Padding attacks**

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. As of February 2013, TLS implementors were still working on developing fixes to protect against this form of attack.

- **RC4 attacks**

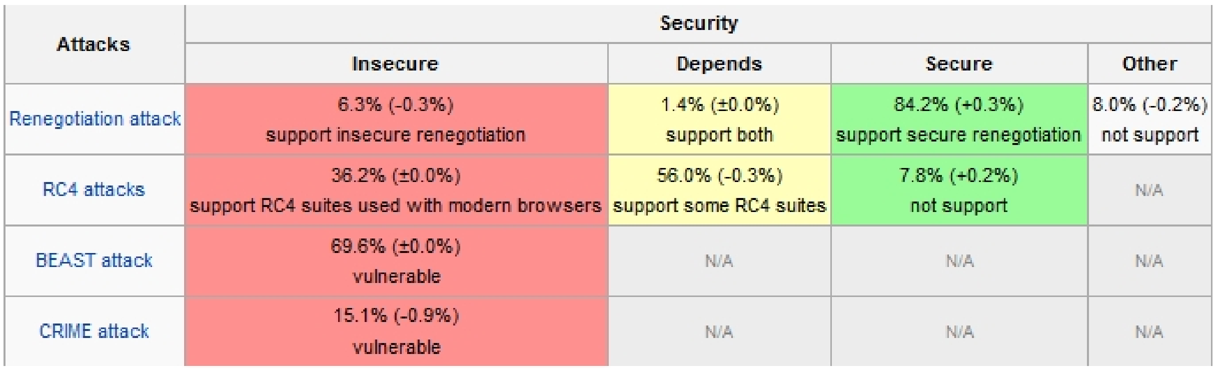
In spite of existing attacks on RC4 that break it, the cipher suites based on RC4 in SSL and TLS were considered secure because of how the cipher was used in these protocols. In 2011 RC4 suite was actually recommended as a work around for the BEAST attack. In 2013 however there was an attack scenario proposed by AlFardan, Bernstein, Paterson, Poettering and Schuldt that uses newly discovered statistical biases in RC4 key table to recover parts of plaintext with large number of TLS encryptions. A double-byte bias attack on RC4 in TLS and SSL that requires 13×2^{20} encryptions to break RC4 was unveiled on 8 July 2013, and it was described as "feasible" in the accompanying presentation at the 22nd USENIX Security Symposium on August 15, 2013. Microsoft recommends disabling RC4 where possible.

➢ Truncation attack

A TLS 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.

Published in July 2013, 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).

Survey of the TLS vulnerabilities of the most popular websites



Attacks	Security			
	Insecure	Depends	Secure	Other
Renegotiation attack	6.3% (-0.3%) support insecure renegotiation	1.4% (±0.0%) support both	84.2% (+0.3%) support secure renegotiation	8.0% (-0.2%) not support
RC4 attacks	36.2% (±0.0%) support RC4 suites used with modern browsers	56.0% (-0.3%) support some RC4 suites	7.8% (+0.2%) not support	N/A
BEAST attack	69.6% (±0.0%) vulnerable	N/A	N/A	N/A
CRIME attack	15.1% (-0.9%) vulnerable	N/A	N/A	N/A

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. 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. However, many web servers providing TLS are not configured to implement such restrictions. 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). 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.

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.

TLS record - This is the general format of all TLS records.

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	Content type			
Bytes 1..4	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
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application

Version - This field identifies the major and minor version of TLS for the contained message. For a ClientHello message, this need not be the *highest* version supported by the client.

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

Length - The length of Protocol message(s), 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, 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 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 for signalling that these parameters will take effect in all further records sent by the same peer.

ADDITIONAL INFO:

Transport Layer Security (TLS) concepts

The Transport Layer Security (TLS) protocol enables two parties to communicate with privacy and data integrity. The TLS protocol evolved from the SSL 3.0 protocol but TLS and SSL do not interoperate.

The TLS protocol provides communications security over the internet, and allows client/server applications to communicate in a way that is private and reliable. The protocol has two layers: the TLS Record Protocol and the TLS Handshake Protocol, and these are layered above a transport protocol such as TCP/IP.

The TLS protocol evolved from the Netscape SSL 3.0 protocol. Although similar, TLS and SSL are not interoperable.

The TLS protocol applies when any of the following CipherSpecs are specified:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_DES_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_NULL_MD5
- TLS_RSA_WITH_NULL_SHA
- TLS_RSA_WITH_NULL_SHA256
- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_RSA_WITH_RC4_128_MD5
- TLS_RSA_WITH_RC4_40_MD5

For more information about the TLS protocol, see the information provided by the TLS Working Group on the web site of the Internet Engineering Task Force at http://www.ietf.org.

For more information see:

http://en.wikipedia.org/wiki/Transport_Layer_Security

http://www.ietf.org/rfc/rfc2246.txt