Security Protocols

- Properties of a security protocol
- SSL/TLS

Goals with this lecture

The most important things you should learn are:

- What characterizes a secure protocol
- How TLS works and its main features
- How ciphers and crypto-keys are negotiated
- What security problems TLS addresses

- Most of these techniques come back in other protocols
  - SSH, IPsec, ...

Cryptographic Protocols

Application layer	• SSL/TLS (IETF) • SSH (IETF)
Network layer	• IPSec (IETF)
Link layer	• WLAN (IEEE 802.11) • MACSec (IEEE 802.11ae)
Physical layer	

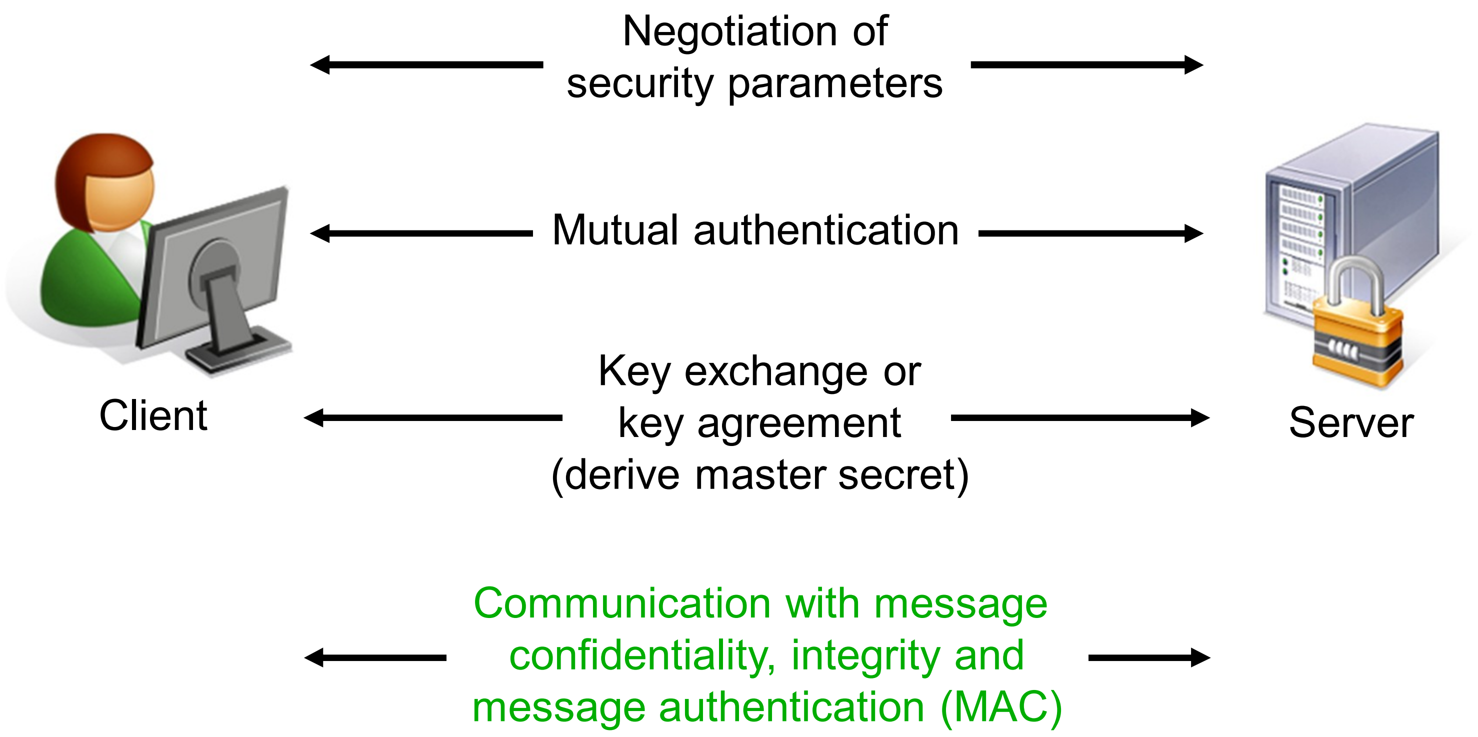
Properties for security protocols

- Data confidentiality
  - Encryption with symmetric keys (AES, …)
- Data integrity
  - Prevent modification of data (keyed hashes, HMAC)
- Data authenticity
  - Guarantees data can not be inserted, deleted, reordered or replayed (can be argued that this should be covered by data integrity)
  - Freshness guarantees (time-stamps, nonces)
- Mutual authentication of communicating parties
  - I requested, both parties know who they talk to (PSK, Certificates)
- Perfect forward secrecy, PFS
  - The transmission (i.e. session keys) should not be revealed if older or future session keys are revealed
  - Nor if user’s pre-shared, public or private keys are compromised (Diffie-Hellman, …)

Perfect Forward Secrecy (PFS)

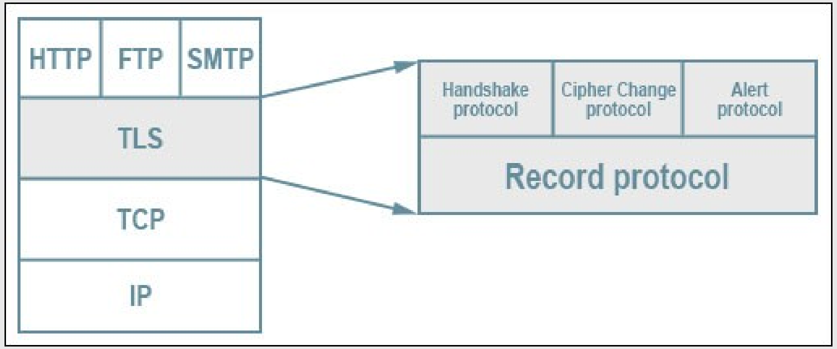
- *Forward Secrecy* = a compromised session key should not affect other sessions (in the past or in the future)
  - Therefore, all sessions must have unique session keys
- *<u>Perfect</u> Forward Secrecy* = the key will not be compromised even if other keys derived from the same long-term keying material are compromised
  - E.g. if session keys are regularly changed and calculated from the same master secret
- Session keys should not depend on a shared secret or private/public keys used for authentication (e.g. RSA)
  - If they do, session keys for other sessions can be derived
  - Use Diffie-Hellman (D-H) or similar, to create unique session keys
- D-H does not protect against MITM attacks – no authentication
  - Use D-H with RSA (DHE_RSA) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)
  - Ephemeral = short lived (one negotiation per handshake) → forward secrecy
  - Together they offer both authentication and unique session keys
- Sometimes PFS is sacrificed for lack of computational resources (IoT devices)
  - ECDH can be used (with static/fixed DH keys)

Typical Cryptographic Protocol



The SSL/TLS protocol

Chapter 17.1-3



SSL/TLS

SSL – **S**ecure **S**ockets **L**ayer

- Developed by Netscape
- Version 1 was for internal use (1994)
- Version 2 incorporated in their “Navigator” web browser (1995).
  Had some problems with MITM attacks
- Version 3 was created with public review from industry (1996)
- IETF standardized SSL version 3.1 and renamed it “TLS version 1.0” (1999)

TLS – **T**ransport **L**ayer **S**ecurity (RFC 2246)

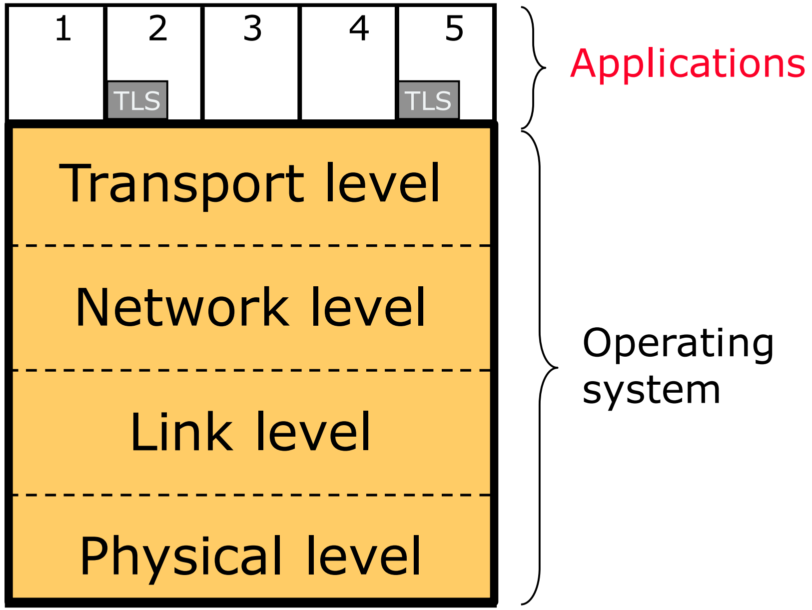
- TLS 1.0 (1999) and 1.1 (2006) – vulnerable and unsupported by web browsers (March 2020)
- TLS 1.2 (2008) with improved hash functions and ECC support
  - RFC 5246
  - Required for HTTP/2 (if TLS is used) (2015)
- TLS 1.3 (2018) redesigned from scratch
  - Took five years of development and testing
  - Faster, better privacy with encrypted handshake with fewer options, better ciphers and modes
  - Faster negotiation of ciphers (fewer RTT)
  - RFC 8446

SSL/TLS

- Designed to protect TCP traffic for applications
  - Applications must be TLS enabled by design
  - TLS can be used to secure all types of TCP connections
- Typical applications:
  - Secure http (https) in web browser uses SSL/TLS – port 443 instead of 80
  - SSL/TLS versions of known protocols: telnets, nntps, imaps, smtps, ldaps, https, …
- Clients authenticate servers when connecting
  - Certificates sent from server to client
  - May (should) also consult CRLs (certificate revocation lists)
- Mutual authentication with client certificates supported – not used by https
- TLS extensions can be included in the first Hello message (similar to TCP options in the TCP three way handshake)
  - E.g. padding, encrypt_then_mac, enable heartbeats [RFC 6520]
- This presentation is not a full description of all features in the protocol – the full protocol specification (in RFCs) is rather long

TLS is integrated in applications

- TLS is used by applications to secure their communications
- Each application runs in its own protected memory
  - They can not see each other
  - They need to do system calls and ask for a network service from the operating system
- Most TLS-enabled applications use a standard library containing all TLS functionality
  - The library becomes part of the application
  - Many free libraries available (OpenSSL, WolfSSL, mbed TLS, …)
- The alternative would be to implement all crypto-functions yourself with your own personal bugs…



OpenSSL
Cryptography and SSL/TLS Toolkit

Comparison of TLS implementations:
https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations

Welcome to OpenSSL!

The OpenSSL Project develops and maintains the OpenSSL software - a robust, commercial-grade, full-featured toolkit for general-purpose cryptography and secure communication. The project's technical decision making is managed by the OpenSSL Technical Committee (OTC) and the project governance is managed by the OpenSSL Management Committee (OMC). The project operates under formal Bylaws.

CVE-2023-0465 Invalid certificate policies in leaf certificates are silently ignored [Low severity] 23 March 2023

CVE-2023-0466 Certificate policy check not enabled [Low severity] 21 March 2023

CVE-2023-0464 Excessive Resource Usage Verifying X.509 Policy Constraints [Low severity] 21 March 2023

CVE-2023-0401 NULL dereference during PKCS7 data verification [Moderate severity] 07 February 2023

CVE-2023-0286 X.400 address type confusion in X.509 GeneralName [High severity] 07 February 2023

CVE-2023-0217 NULL dereference validating DSA public key [Moderate severity] 07 February 2023

CVE-2023-0216 Invalid pointer dereference in d2i_PKCS7 functions [Moderate severity] 07 February 2023

CVE-2023-0215 Use-after-free following BIO_new_NDEF [Moderate severity] 07 February 2023

CVE-2022-4450 Double free after calling PEM_read_bio_ex [Moderate severity] 07 February 2023

CVE-2022-4304 Timing Oracle in RSA Decryption [Moderate severity] 07 February 2023

CVE-2022-4203 X.509 Name Constraints Read Buffer Overflow [Moderate severity] 07 February 2023

```
X.400 address type confusion in X.509 GeneralName (CVE-2023-0286)
=================================================================

Severity: High

There is a type confusion vulnerability relating to X.400 address processing
inside an X.509 GeneralName. X.400 addresses were parsed as an ASN1_STRING but
the public structure definition for GENERAL_NAME incorrectly specified the type
of the x400Address field as ASN1_TYPE. This field is subsequently interpreted by
the OpenSSL function GENERAL_NAME_cmp as an ASN1_TYPE rather than an
ASN1_STRING.

When CRL checking is enabled (i.e. the application sets the
X509_V_FLAG_CRL_CHECK flag), this vulnerability may allow an attacker to pass
arbitrary pointers to a memcmp call, enabling them to read memory contents or
enact a denial of service. In most cases, the attack requires the attacker to
provide both the certificate chain and CRL, neither of which need to have a
valid signature. If the attacker only controls one of these inputs, the other
input must already contain an X.400 address as a CRL distribution point, which
is uncommon. As such, this vulnerability is most likely to only affect
applications which have implemented their own functionality for retrieving CRLs
over a network.

OpenSSL versions 3.0, 1.1.1 and 1.0.2 are vulnerable to this issue.
```

https://www.openssl.org/news/vulnerabilities.html https://www.gnutls.org/

The GnuTLS Transport Layer Security Library

Welcome to *GnuTLS* project pages

- **Overview**
  GnuTLS is a secure communications library implementing the SSL, TLS and DTLS protocols and technologies around them. It provides a simple C language application programming interface (API) to access the secure communications protocols as well as APIs to parse and write X.509, PKCS #12, and other required structures.

  The project strives to provide a secure communications back-end, simple to use and integrated with the rest of the base Linux libraries. A back-end designed to work and be secure out of the box, keeping the complexity of TLS and PKI out of application code.

- **Features**
  - Support for TLS 1.3, 1.2, 1.1, 1.0 protocols, and (optionally) SSL 3.0
  - Support for DTLS 1.2, and DTLS 1.0, protocols
  - Support for certificate path validation, as well as DANE and trust on first use.
  - Support for the Online Certificate Status Protocol (OCSP).
  - Support for public key methods, including RSA and Elliptic curves, as well as password and key authentication methods such as SRP and PSK protocols.
  - Support for all the strong encryption algorithms, including AES and Camellia.
  - Support for CPU-assisted cryptography with VIA padlock and AES-NI instruction sets.
  - Support for cryptographic accelerator drivers via /dev/crypto.
  - Supports natively HSMs and cryptographic tokens, via PKCS #11 and the Trusted Platform Module (TPM).
  - Runs on most Unix platforms and Windows.

- **License**
  The core library licensed under the GNU Lesser General Public License version 2.1 (LGPLv2.1+). The LGPL license is compatible with a wide range of free licenses, and even permit you to use GnuTLS in non-free proprietary programs.

- **Documentation:**
  You can obtain GnuTLS' manual at lulu.com or download any of the electronic formats.

For more information on GnuTLS features, see the wikipedia article comparing different TLS implementations.

News flashes Follow @GnuTLS

2023-02-10	Released GnuTLS 3.8.0 a bug-fix and enhancement release on the 3.8.x branch. Added the GNUTLS-SA-2020-07-14 security advisory.
2022-09-27	Released GnuTLS 3.7.8 a bug-fix release on the 3.7.x branch.
2022-07-28	Released GnuTLS 3.7.7 a bug-fix and enhancement release on the 3.7.x branch. Added the GNUTLS-SA-2022-07-07 security advisory.
2022-05-27	Released GnuTLS 3.7.6 a bug-fix release on the 3.7.x branch.

TLS Protocol Architecture [RFC 5246]

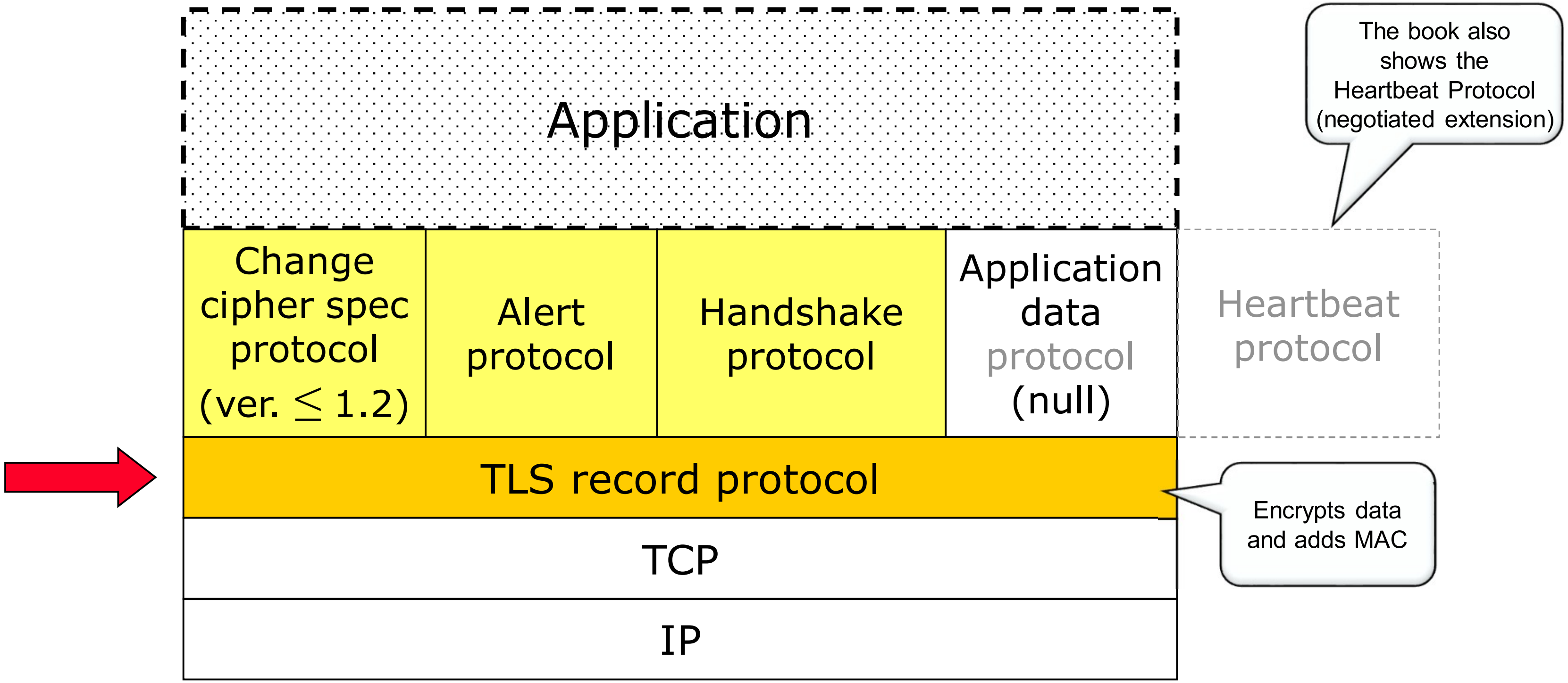
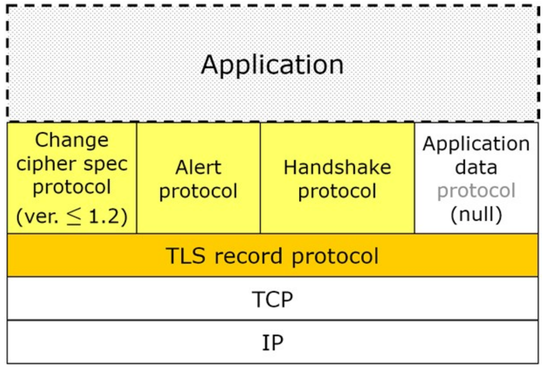


Fig. 17.2

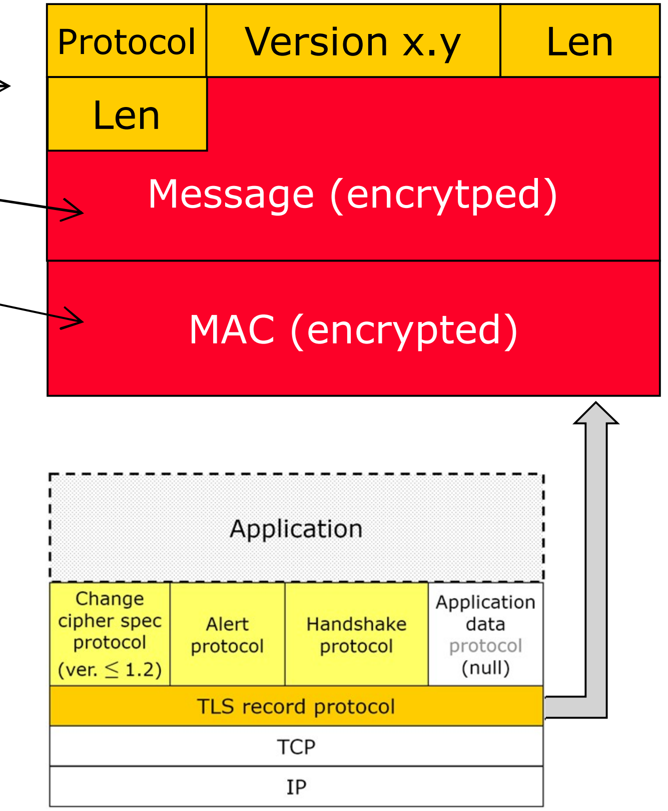
The TLS Record Protocol

- Active after all initial handshaking done – when algorithms and keys are negotiated
- Relies on TCP to offer reliable communication (to do retransmissions)
- Provides confidentiality and message integrity
  - Encrypts data and adds MAC (HMAC, keyed hash)
- May also fragment and compress data



The TLS Record Protocol

- Format:
  - 5-byte header
  - Message (data), max 16,384 bytes optionally compressed
  - Message Authentication Code
- **Protocol** field tells what upper-layer TLS-protocol it encapsulates:
  - 20 = Change cipher protocol
  - 21 = Alert protocol
  - 22 = Handshake protocol
  - 23 = Application (protocol) data
- **Version** = SSL/TLS protocol version
  - 2 bytes: major and minor number
  - SSL = 2.0 or 3.0
  - TLS version 1.0 = 3.1, etc.



The TLS Record Protocol

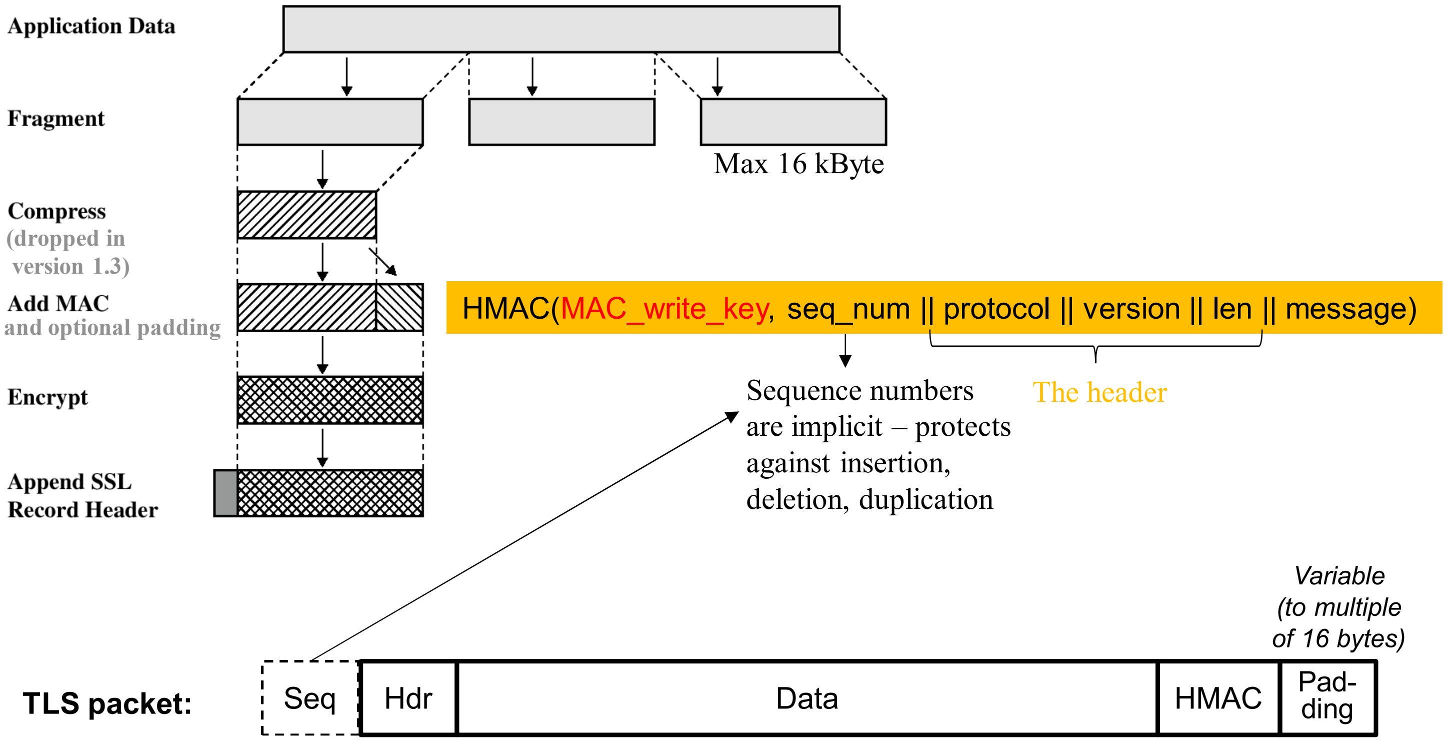
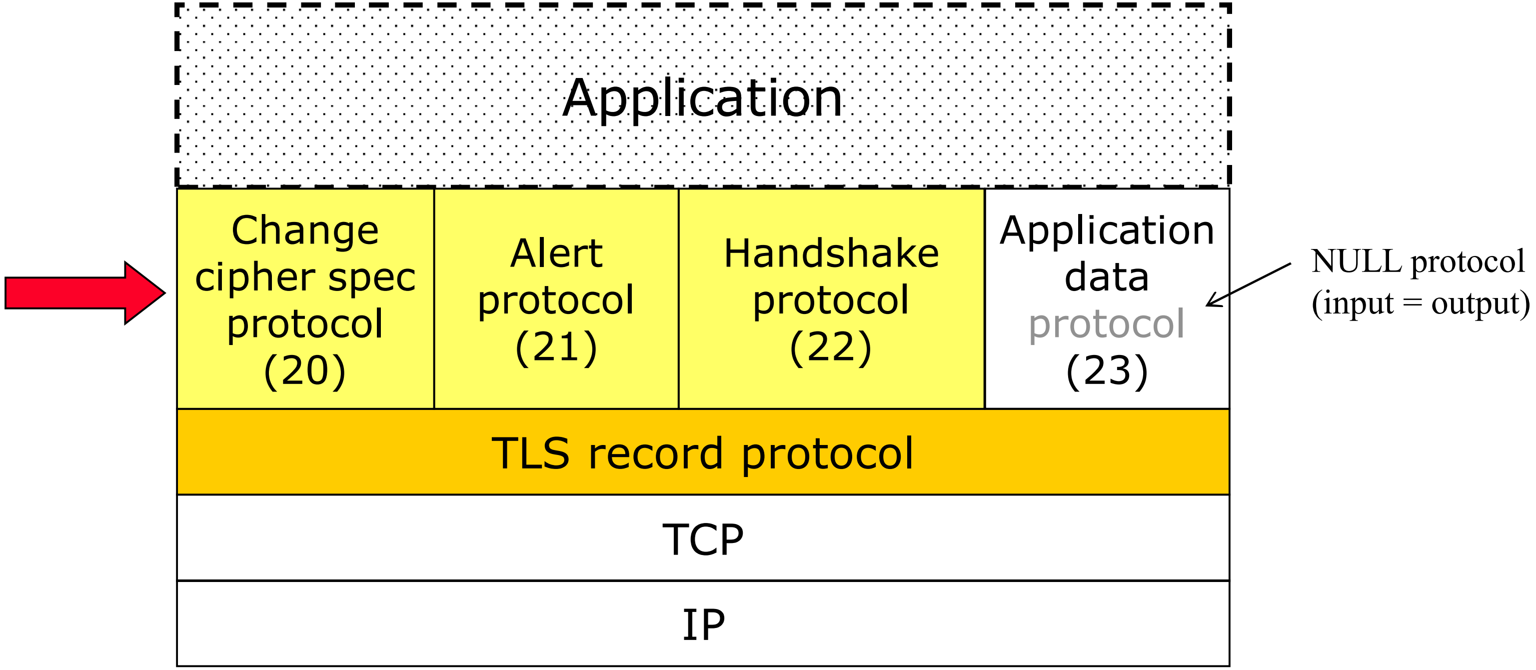
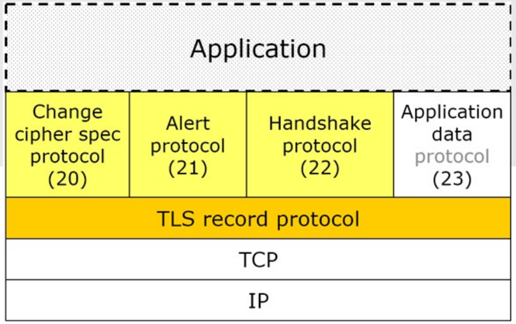


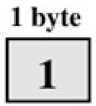
Fig. 17.3

TLS Protocol Architecture

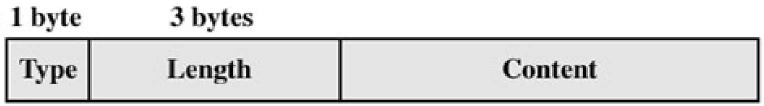


TLS "Protocols"

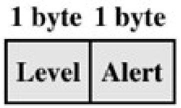




(a) Change Cipher Spec Protocol
(dropped in version 1.3)



(c) Handshake Protocol



(b) Alert Protocol

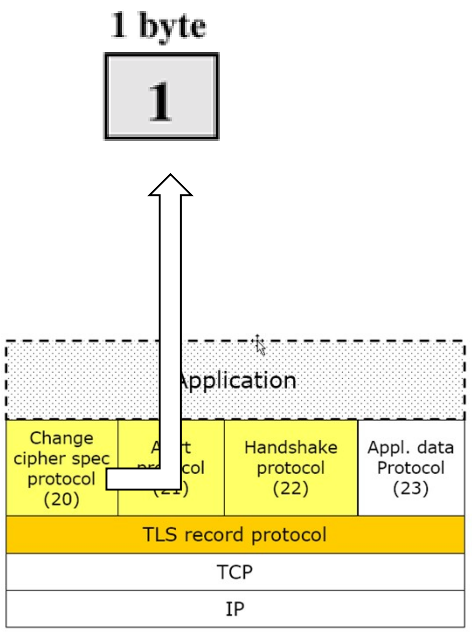
(d) Other Upper-Layer Protocol (e.g., HTTP)

(Application data)

Fig. 17.5

The TLS Change Cipher Spec Protocol

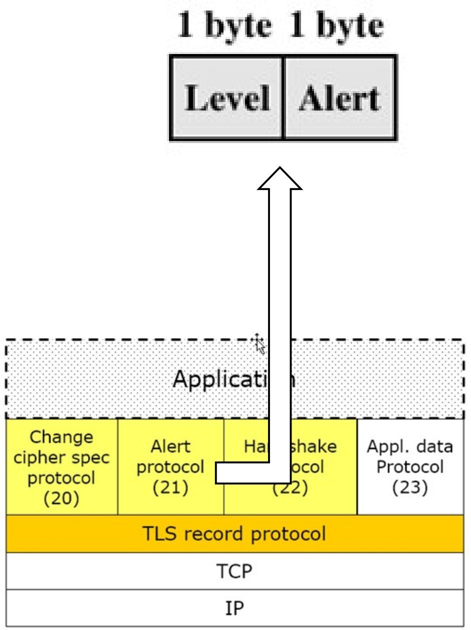
- Just a one-byte message: **1**
- Pending state becomes current state
  - Changes encryption to what has been negotiated earlier (algorithms, keys)
- Depreciated in TLS ver. 1.3
  (may be sent but is ignored)



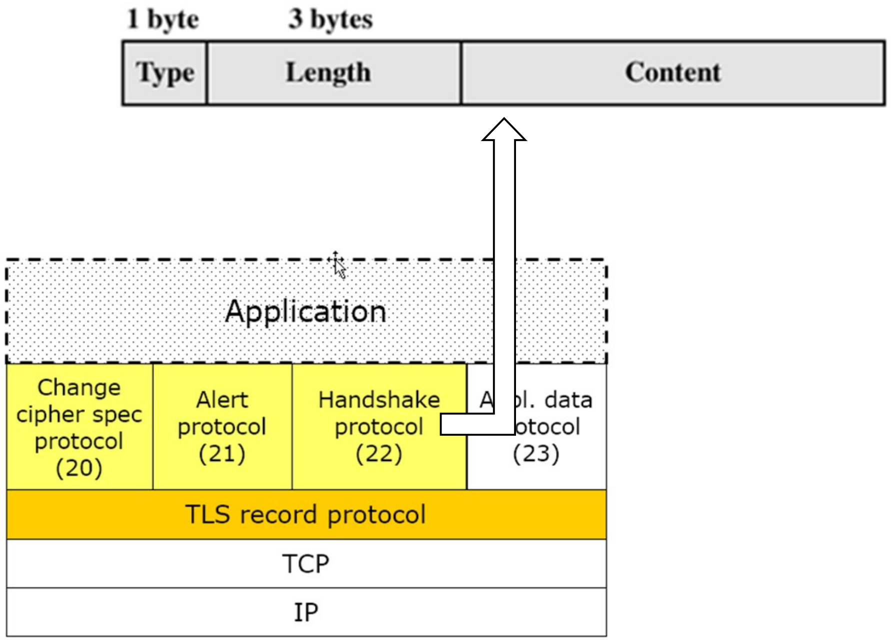
The TLS Alert Protocol

Signals error and alert conditions:

- Severity Level: warning or fatal
  - Must immediately disconnect if fatal
- Alert codes (SSL has 12, TLS twice as many):
  - CloseNotify
  - UnexpectedMessage (fatal)
  - BadRecordMAC (fatal)
  - Certificate Unsupported/Revoked/Expired/Unknown/Bad
  - Illegal Parameter
  - HandshakeFailure (not possible to agree on services, fatal)
  - …



SSL/TLS handshake protocol



Algorithm (cipher) negotiation

- Authentication method
- Data encryption algorithm
- Data protection algorithm
- Performs key exchange

Algorithms to negotiate

- Authentication + session key exchange
  - **PSK** (pre-shared keys) mixed with DH to get forward secrecy
  - **RSA** (public keys) – same here, DH must also be used (with short-lived keys)
  - Fixed Diffie-Hellman: certificate contains pre-calculated primes (groups)
  - (Ephemeral) Diffie-Hellman: each side generates one-time parameters
  - ECDHE = Elliptic curve Diffie-Hellman guarantees forward secrecy
- Cipher for data encryption
  - Weak ciphers: DES_40, DES_56, 3-DES, RC4
  - Weak cipher modes: AES_CBC
  - Ciphers: AES_128, AES_256, …

Weak ciphers only allowed in TLS ≤ 1.2

- MAC algorithm for data integrity (ver. 1.2):
  - Hash functions: MD5, SHA-1, SHA-256
  - $\mathrm{HMAC}_K(\mathrm{msg}) = \mathrm{hash}(K \oplus \mathrm{opad} \,||\, \mathrm{hash}(K \oplus \mathrm{ipad} \,||\, \mathrm{msg}))$
  - Calculation:

opad = 0x5C5C… ipad = 0x3636…

HMAC_K(seq_num || tls_proto || tls_vers || msg_len || msg)

Examples of Cipher Suites (ver. 1.2)

Cipher Suite	Key Negotiation	Digital Signature Method	Symmetric Key Encryption Method	Hashing Method for HMAC	Strength	
NULL_WITH_NULL_NULL	None	None	None	None	None	0x0000
RSA_EXPORT_WITH_RC4_40_MD5	RSA export strength (40 bits)	RSA export strength (40 bits)	RC4 (40-bit key)	MD5	Very weak	0x0003
RSA_WITH_AES_256_CBC_SHA256	RSA	RSA	AES 256 bits	SHA-256	Weak, no PFS	0x0069
DHE_RSA_WITH_AES_128_GCM_SHA256	DH+RSA	RSA	AES 128 bits	SHA-256	Strong	0x009E

https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

Protocol support in Chrome, 112

Protocol Features

Protocols

TLS 1.3	Yes
TLS 1.2	Yes
TLS 1.1	No
TLS 1.0	No
SSL 3	No
SSL 2	No

Cipher Suites (in order of preference)

TLS_GREASE_2A (0x2a2a)	-
TLS_AES_128_GCM_SHA256 (0x1301) Forward Secrecy	128
TLS_AES_256_GCM_SHA384 (0x1302) Forward Secrecy	256
TLS_CHACHA20_POLY1305_SHA256 (0x1303) Forward Secrecy	256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b) Forward Secrecy	128
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) Forward Secrecy	128
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c) Forward Secrecy	256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) Forward Secrecy	256
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9) Forward Secrecy	256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8) Forward Secrecy	256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) WEAK	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) WEAK	256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c) WEAK	128

https://www.ssllabs.com/ssltest/viewMyClient.html

Qualys. SSL Labs

Home | Projects | Qualys Free Trial | Contact

You are here: Home > Projects > SSL Server Test > williamstallings.com

SSL Report: williamstallings.com (209.237.150.20)

Assessed on: Mon, 06 Mar 2023 11:43:58 UTC | Hide | Clear cache

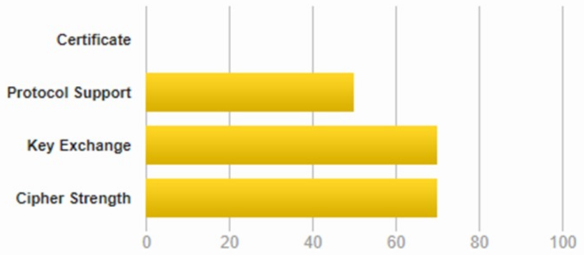
Scan Another »

Summary

Overall Rating

T

If trust issues are ignored: C



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server's certificate is not trusted, see below for details.

This server supports weak Diffie-Hellman (DH) key exchange parameters. Grade capped to B. MORE INFO » — Logjam attack

The server supports only older protocols, but not the current best TLS 1.2 or TLS 1.3. Grade capped to C. MORE INFO »

This server does not support Forward Secrecy with the reference browsers. Grade capped to B. MORE INFO » — RSA + DH but not short-lived keys

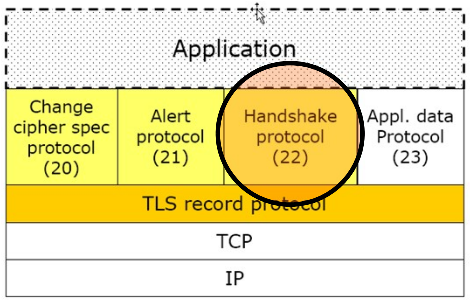
This server does not support Authenticated encryption (AEAD) cipher suites. Grade capped to B. MORE INFO » — TLS CBC timing attack

This server supports TLS 1.0. Grade capped to B. MORE INFO »

Transport Layer Security (TLS)

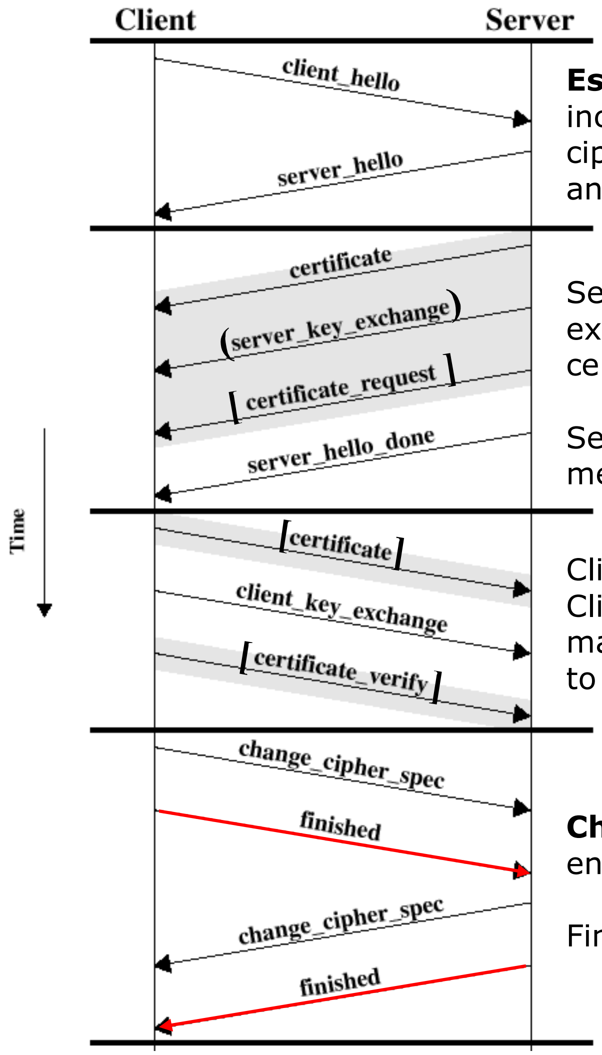
(version 1.2 and earlier)

SSL/TLS handshake protocol



Notes:
- Client certificates are not commonly used, thus messages within [ ] are normally not sent

- Only messages in red are encrypted



Establish security capabilities, incl. protocol version, session ID, cipher suite, compression method and initial random numbers

Server **may send certificate**, key exchange parameters and a certificate_request.

Server signals end of hello message phase.

Client sends certificate if requested. Client sends **key_exchange**. Client may send certificate_verification to show it knows the private key.

Change_cipher_spec turns on encryption.

Finish ends the handshake protocol

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Fig. 17.6

Handshake Round 1

Client hello Client $\xrightarrow{\{\, ver_C \,\|\, r_c \,\|\, sid \,\|\, ciphers \,\|\, comps \,\}}$ Server

Client $\xleftarrow{\{ ver \,\|\, r_s \,\|\, sid \,\|\, cipher \,\|\, comp \,\}}$ Server **Server hello**

ver_C	Highest version of protocol client wants to use
r_c, r_s	nonces (4 byte timestamp and 28 random bytes) – seed for encryption
sid	Current session id, 32 bytes long (0 if new session)
ciphers	<u>List</u> of ciphers that client understands (preference order)
comps	List of compression algorithms that client understands
ver	Version of protocol to be used (highest version both understand)
cipher	Cipher to be used (selected from client's list)
comp	Compression algorithm to be used

Some notes about round 1

- The 32-byte nonces r_c and r_s contain two things:
  - Time stamp (4 bytes) to guarantee each nonce is unique
  - Random number (28 bytes)
  - Used when calculating keys

Time stamp	………… random number …………

- Random numbers (general remark):
  - Must be created by a strong (i.e. cryptographic) random number generator
  - If not random, attackers can try to pre-compute values
  - Both server and client are responsible in creating random numbers (r_c and r_s)
- The most common key negotiation cipher used by web browsers is RSA
- Session ID:
  - Always zero for a new session (or long random string which is ignored)
  - Server responds with a session_ID > 0 if it allows sessions to be resumed
  - Makes it possible to use keying material from a previously established session
  - Faster since public key operations are time consuming

The Client Hello message

```
⌄ Transport Layer Security
  ⌄ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 512
    ⌄ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 508
        Version: TLS 1.2 (0x0303)
      › Random: abf1fac49409cbaef37bf577e3b45fd61b3dec375c456ed8…
        Session ID Length: 32
        Session ID: d7e938ee24deb6fd30c827b4e4cc5f1205567afc125c601d…
        Cipher Suites Length: 34
      ⌄ Cipher Suites (17 suites)
          Cipher Suite: Reserved (GREASE) (0x1a1a)
          Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
          Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
          Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)   ← To be chosen by server…
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
           …
       ⌄ Compression Methods (1 method)
           Compression Method: null (0)
```

nonce r_c

Ciphers supported by the client. Listed in order of preference

The Server Hello message (from Google)

```
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 89
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 85
      Version: TLS 1.2 (0x0303)
      Random: f373e985c9a960e175a66ba6c0fc2a893fccff79e3518f44…
      Session ID Length: 32
      Session ID: 7fb1a61d097f4f2f8cb658e3e3a9a9a7071790a1e7ceefc8…
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Compression Method: null (0)
      Extensions Length: 13
      Extension: renegotiation_info (len=1)
      Extension: ec_point_formats (len=4)
```

nonce r_s

Selected cipher

Handshake Round 2

If TLS_DHE: Ephemeral D-H key exchange, server selects (p, g) and calcualtes $Y = g^b \bmod p$.

Client ← $\{$ X.509 *certificate (one or a chain)* $\}$ — Server **certificate**

Client ← $\{ p \parallel g \parallel Y_s \parallel sign(k_S, r_c \parallel r_s \parallel p \parallel g \parallel Y_s) \}$ — Server **Server key exchange**

Client ← $\{ ctype \parallel gca \}$ — Server **Certificate request**

Client ← $\{ \}$ — Server **Hello done**

Contents of second message is algorithm dependent – here ephemeral D-H

p, g	Modulo and generator to use: $Y = g^b \bmod p$ where "b" is secret
k_S	Server key to sign hash – algorithm from negotiation (DSS, RSA, …)
ctype	Certificate type requested from client, if any
gca	List of names of acceptable certification authorities (helps client to chose)

Handshake Round 3

If requested by server. Contains client's public key

certificate Client $\{\ client\_cert\ \}$ Server

If RSA: 48 random bytes encrypted with server's public key k_p

Client key exchange Client D-H: $\{\ Y_c\ \}$; RSA: $\{\ pre\_master\_secret\ \}k_p$ Server

If D-H: client parameters

Only if client has sent a certificate. Shows it is in possession of the private key k_c.

Certificate verify Client $\{\ sign(k_c,\ master\_secret,\ msgs)\ \}$ Server

msgs	Concatenation of messages sent/received so far
ipad	0x3636… repeated to block length
opad	0x5c5c… repeated to block length
k_c	Client's private key
Master_secret	Calculated master secret (calculated from pre_master_secret)

The Pseudo-random function [RFC 5246]

- A pseudo-random function is defined and used in TLS:

```
PRF(k, label, x) =
    HMACk(HMACk(label||x) || label||x) ||
    HMACk(HMACk(HMACk(label||x)) || label||x) ||
    HMACk(HMACk(HMACk(HMACk(label||x))) || label||x) ||
    …
```

- SHA-256 gives 32 bytes per round/hash
- Used to expand a short secret to arbitrary length blocks

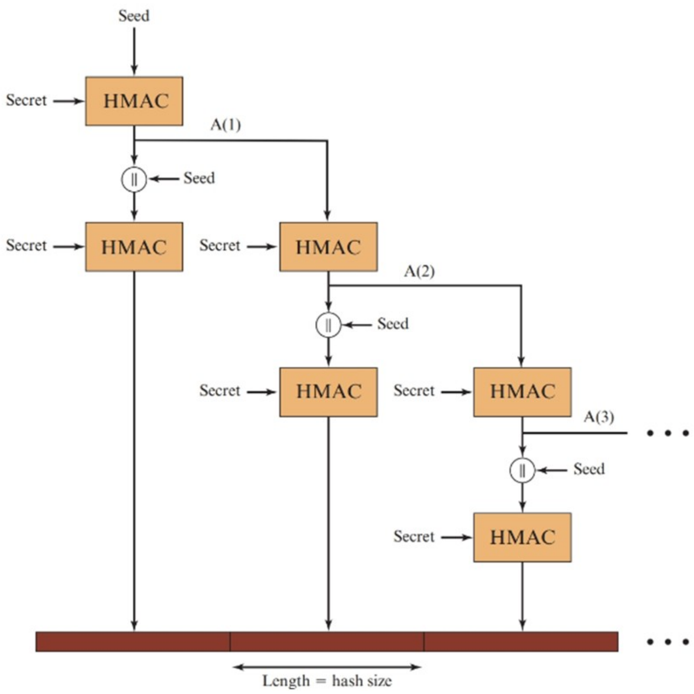


Fig 17.7

1 Pre-Master Secret → Master Secret

- If RSA server authentication (with certificates) then:
  - Client generates a ***pre_master_secret***, encrypts it with the server's public key and sends it to the server (48 random bytes)
  - Only the correct server can make use of it
  - Offers protection against MITM attacks but no PFS (problems if server's private key becomes known)
- Diffie-Hellman can (should) be used to calculate the ***pre_master_secret***
  - MITM-attacks are possible so should be combined with RSA or ECC Signatures (ECDSA)
  - Offers Forward Secrecy
- The *master secret* can now be calculated:

$$\underbrace{\textit{master\_secret}}_{} = \mathbf{PRF}(\underbrace{\textit{pre\_master\_secret}}_{k}, \underbrace{\text{“}\texttt{master secret}\text{”}}_{label}, \underbrace{r_c \,||\, r_s}_{x})$$

- Result: a 48 byte long master secret based on data that both the server and the client have generated

2 Master secret → Encryption keys

- An arbitrarily long stream of keys can now be created from the master secret:

  key_material = PRF(master_secret, `"key expansion"`, $r_c \,||\, r_s$);

- The PRF is used whenever more key material is needed
- From the key material 6 different keys are created:
  - Write keys are used to encrypt data
  - MAC keys for integrity protection
  - IV for cipher if CBC mode used (only generated if needed)
- We have one key for each direction
- We never use the master secret directly
  - Used only to generate keys that are frequently changed

Server write key

Client write key

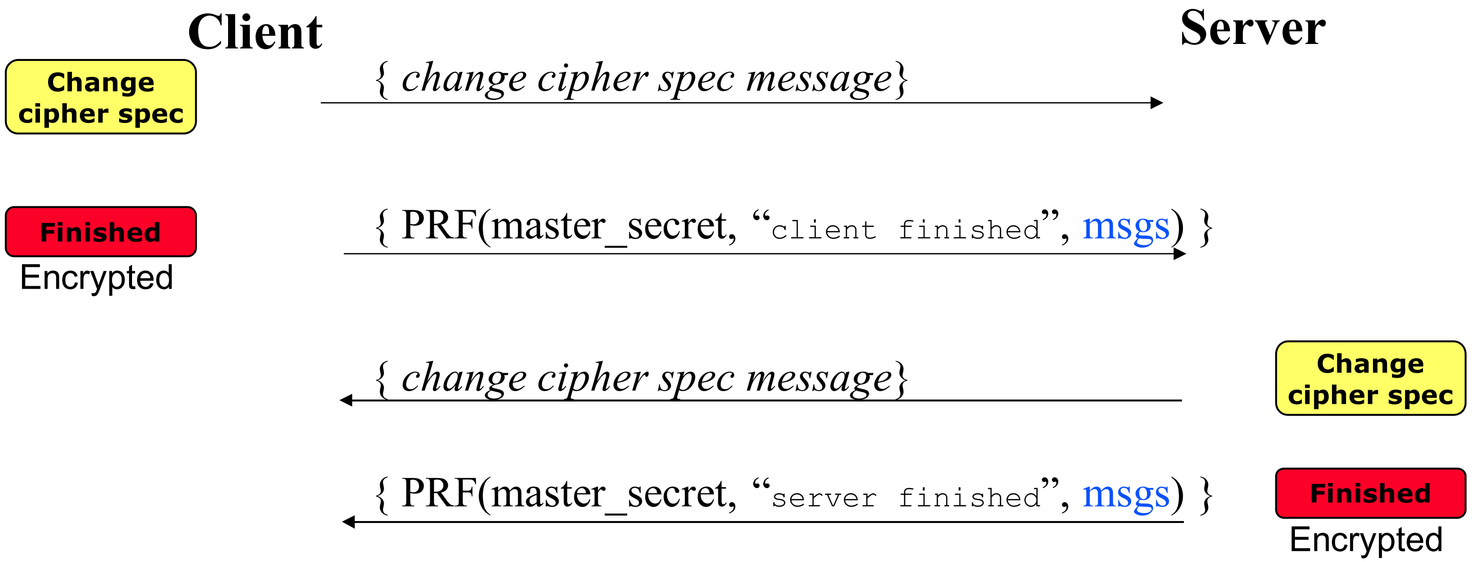
Server MAC key

Client MAC key

Server IV

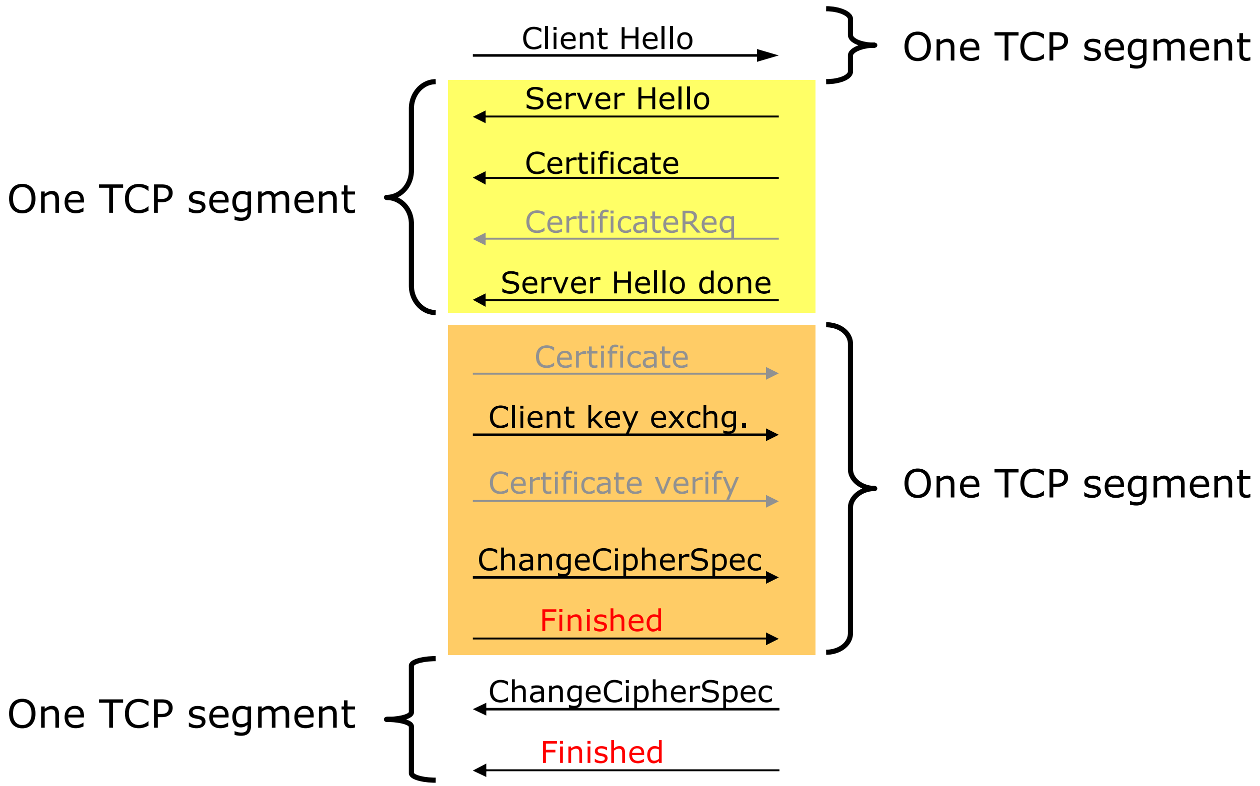
Client IV

Handshake Round 4



msgs = hash(all_handshake_messages_sent)
The hash function to use was negotiated in the hello msgs

TLS rounds – from a TCP perspective



- Two round-trip delays
  - https will be slower than http ☹
  - IoT devices must use more power
- TLS 1.3 addresses this with a new design

Closing a TLS session

- TLS has a procedure to close a connection
  - Sends a “close_notify” message before TCP FIN
  - Rarely used for Web sessions
- Protects against “truncation attacks”
  - An attacker may be able to close down the TCP connection prematurely
  - If a transaction consists of several messages, the last part may otherwise be lost
- Alternative: handle it in the application-level protocol
  - TCP has no idea

Transport Layer Security (TLS) Version 1.3

https://tools.ietf.org/html/rfc8446

160 pages

(Book only lists features of ver. 1.3 on the last page of chapter 17.2)

Problems
in earlier versions

Patching, patching
and patching…

This has mitigated quite a few attacks..

OWASP
The Open Web Application Security Project

RC4
- Roos's Bias 1995
- Fluhrer, Martin & Shamir 2001
- Klein 2005
- Combinatorial Problem 2001
- Royal Holloway 2013
- Bar-mitzvah 2015
- NOMORE 2015

RSA-PKCS#1 v1.5 Encryption
- Bleichenbacher 1998
- Jager 2015
- DROWN 2016

Renegotiation
- Marsh Ray Attack 2009
- Renegotiation DoS 2011
- Triple Handshake 2014

3DES
- Sweet32

AES-CBC
- Vaudenay 2002
- Boneh/Brumley 2003
- BEAST 2011
- Lucky13 2013
- POODLE 2014
- Lucky Microseconds 2015

Compression
- CRIME 2012

MD5 & SHA1
- SLOTH 2016
- SHAttered 2017

15

https://www.owasp.org/images/9/91/OWASPLondon20180125_TLSv1.3_Andy_Brodie.pdf

TLS 1.3 released 2018 (RFC 8446)

- Took 5 years of work and testing, 28 drafts…
- Many smaller changes:
  - PRF -> HKDF (HMAC-based Extract-and-Expand Key Derivation Function)
  - New ciphers and signature algorithms added
- Unsafe and unused functions removed – fewer functions are more secure
  - Compression deleted – may reveal information about payload [CRIME attack]
  - 3-DES, RC4, MD5, SHA-1
  - Cipher Block Chaining Mode (AES-CBC) timing leaks [Beast and Lucky13]
  - Export ciphers [Logjam, Freak, Drown, Beast]
  - Static and weak D-H groups
  - …
- Perfect Forward Secrecy mandatory no longer optional
  - Most RSA methods dropped – offered no PFS + hard to implement correctly [Million-message and Robot]
- Faster handshake: **1-RTT** where client guesses ciphers to be used
  - Fewer cipher suites supported – easier to guess what to use
  - If server cannot support request, a new message exist: *HelloRetryRequest*
  - Everything after *Server Hello message* is encrypted
  - Change Cipher Spec protocol not needed
- **Also supports 0-RTT – session resumption with stored information**
  - Less secure – inspired by the QUIC protocol
  - If client and server has communicated before, sessions can be resumed with stored “session tickets”

Security protocols must be updated to have protection against new technology findings

Note that TLS 1.2 is still widely used and considered secure

https://www.thesslstore.com/blog/tls-1-3-everything-possibly-needed-know/
https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/

Key exchange algorithms in TLS 1.3

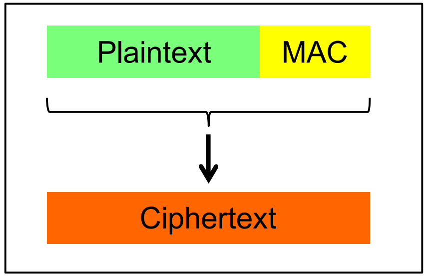
Key exchange/agreement and authentication

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[72]	
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)[73]	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	

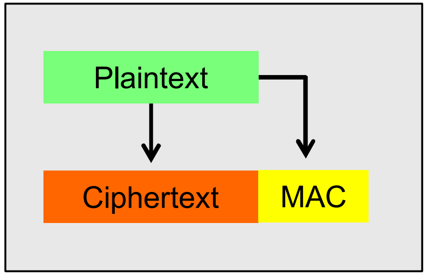
Wikipedia

Order of Encryption and MAC matters?

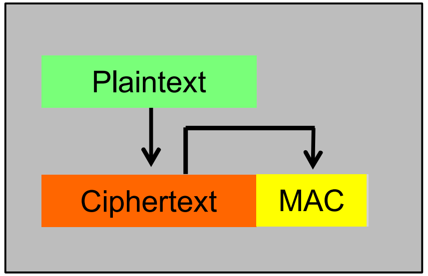
- **MAC-then-Encrypt**
  - Append a keyed MAC to the plaintext and encrypt
  - Only plaintext integrity
  - Receiver must decrypt message before MAC can be checked
  - Padding Oracle Attacks* possible: observe timing and behavior
  - **TLS version 1.2**



- **Encrypt-and-MAC**
  - Encrypt the plaintext, append a keyed MAC of the plaintext
  - Only plaintext integrity,
  - Receiver must decrypt message before MAC can be checked
  - Padding oracle attacks possible
  - **SSH**



- **Encrypt-then-MAC**
  - Encrypt plaintext, append a keyed MAC of the ciphertext
  - Ciphertext integrity but no plaintext integrity
  - If MAC ok, any text produces a cleartext (theoretical problem)
  - Receiver cannot be fed with invalid ciphertexts – good!
  - **IPsec, TLS 1.3**



*time to check MAC leaks information, even worse if packet does not make sense (invalid padding) where MAC is not checked at all

Message authentication in TLS 1.3

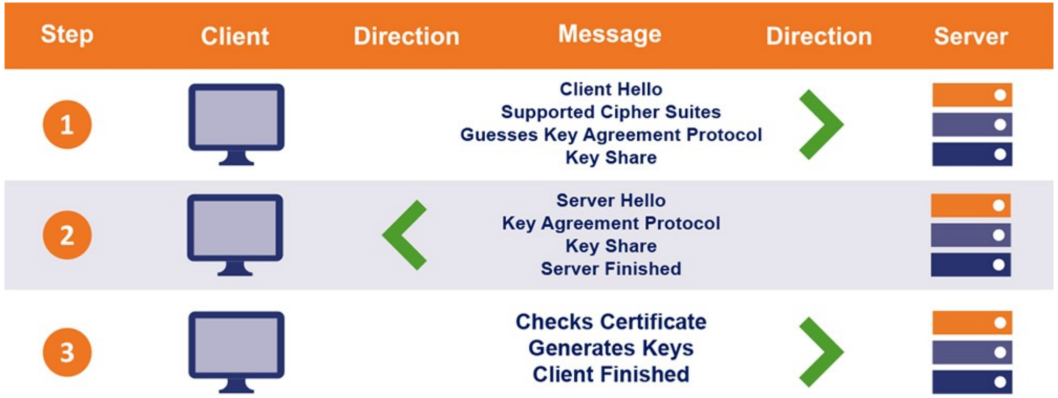
Data integrity

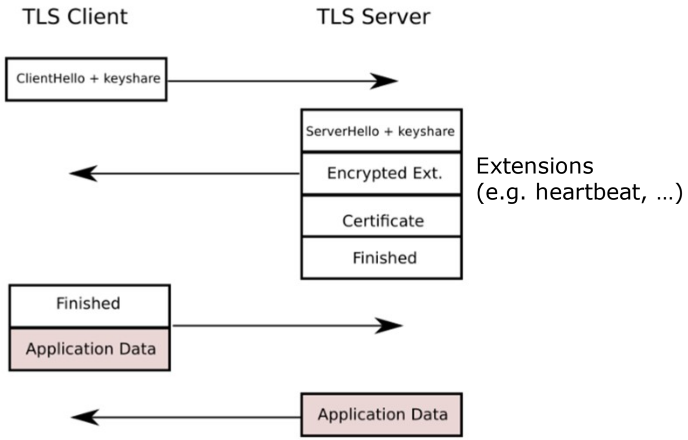
Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
HMAC-MD5	Yes	Yes	Yes	Yes	Yes	No	Defined for TLS 1.2 in RFCs
HMAC-SHA1	No	Yes	Yes	Yes	Yes	No	
HMAC-SHA256/384	No	No	No	No	Yes	No	
AEAD	No	No	No	No	Yes	Yes	
GOST 28147-89 IMIT[74]	No	No	Yes	Yes	Yes	?	Proposed in RFC drafts
GOST R 34.11-94[74]	No	No	Yes	Yes	Yes	?	

- AD = Authenticated data
  - Prevents cut-and-paste attacks of valid ciphertexts into other contexts
  - AD can be any data but should tell where data belongs
  - TLS has sequence numbers, so AD=MAC
- **AEAD** (Authenticated Encryption with associated data)
  - Alternative to the X-then-Y algorithms (previous slide)
  - **Combines** MAC and encryption, gives provable security guarantees
  - Offers confidentiality, integrity, and authenticity
- `AE`$_k$`(IV, message, data)` → `ciphertext + MAC`
  - Example of algorithms: AES_128_GCM
- Also addresses the problem with getting Encrypt-then-MAC right which has been problematic
  - Some research suggest that Encrypt-then-MAC is stronger than AEAD – if implemented certain ways

TLS 1.3 with 1-RTT

- Client Hello message contain all necessary information:
  - List of supported cipher suites
  - A guess of key agreement protocol to use and necessary info to generate keys (Key Share)
- Server responds with key agreement protocol + Key Share/certificate





https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/

TLS 1.3 with 0-RTT (session resumption)

- A further optimized method – although less secure
- Session resumption
  - A resumption master ticket is created known by both parties
  - Client tells "session ID" at resumption
  - Can be kept by client only: it's encrypted by server and sent back when needed
- Replays possible
  - Compare with UDP
  - Avoid with messages that require state change on the server (commit, …)
  - Captured messages may make sense to server
  - There are ways around this, but use 0-RTT only if necessary

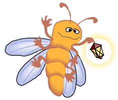
TLS and security

A Detailed Look at RFC 8446 (a.k.a. TLS 1.3)
Describes problems that motivated changes in TLS 1.3

A good summary of various problems in TLS!

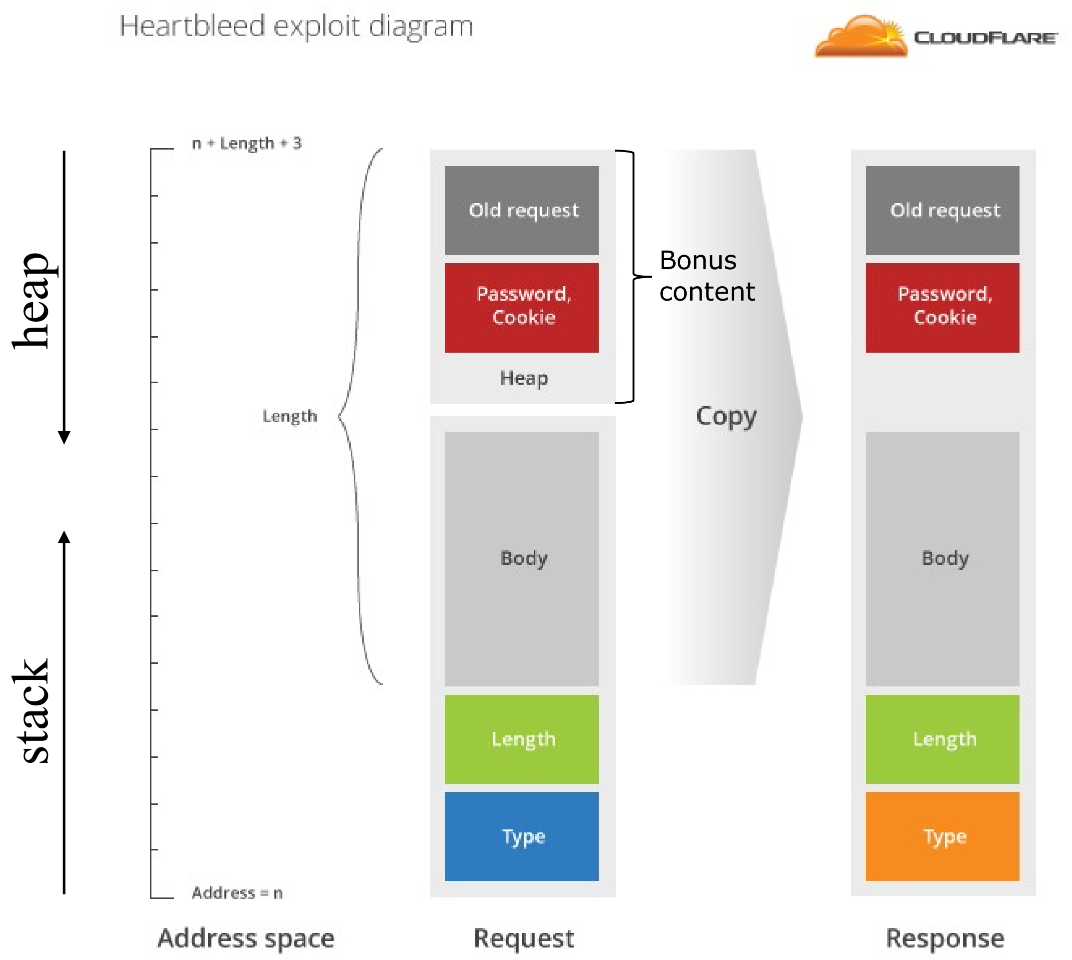
https://blog.cloudflare.com/rfc-8446-aka-tls-1-3

TLS and Security – some notes

- Bugs, bugs, bugs – always the biggest security problem
  
  - Example: If the full certificate chain is not verified, then anyone can sign a certificate
- Poor random number generation
  - Very important with good seeds
- Timing-based crypto-analysis
  
  - Time to decrypt pre-master key may be measured:
    Time between ClientKeyExchg ←→ ChangeCipherSpec
  - May be hard to do timing in real life
  - Countermeasure: add random time or blinding technique (some of the data signed should be unknown to attacker)
  - Padding Oracle Attack can work (see crypto course lab)
- Traffic analysis may reveal size of encrypted user data
  - Size may tell what web page a user accessed
  - Tables can be created for popular web sites
  - Enable random padding in TLS record protocol

The Heartbleed bug

- The Heartbeat protocol is a protocol running on top of the Record Layer (RFC6520)
  - Idea is to check that connection is open
  - And to periodically send a message to make sure firewalls and other equipment does not consider the TCP session closed
- Has two message types:
  - HeartbeatRequest=1 and HeartbeatResponse=2
  - A HeartbeatRequest message can arrive at any time during the lifetime of a connection
  - The receiver echoes back the same message to the sender – trivial protocol!
- A bug is in the OpenSSL's implementation of the TLS heartbeat extension was introduced in OpenSSL March 2012
  - And fixed in OpenSSL 1.0.1g released April 2014
- But there was no check that header length = packet length…



Old contents on the stack and heap will be returned to the client

If lucky (e.g. after reboot) the heap contains private keys

Without Perfect Forward Secrecy, all sessions (older and future) can be decrypted!

https://blog.cloudflare.com/answering-the-critical-question-can-you-get-private-ssl-keys-using-heartbleed

Windows IT Pro

Massive Open Source Internet Security Flaw Affects Two Thirds of All Web Sites

Rod Trent

Apr 9, 2014

Anything that relies on OpenSSL for communication is vulnerable. Here's just a short list example (but it's growing): any Linux-based appliance, routers, Steam, iOS, Android, Mac OS, Smart TVs, DVD/Blu-Ray players, set-top boxes, OpenOffice, Apple Mobile Device Support, BartPE, Trillian, Plesk, ActivePerl, MailEnable, Gene6 FTP, Kindle for PC, IMAPSize, BIND DNS, wput, HP ProLiant System Management and HP Version Control Agent software.

It's being estimated that it may take ten years to clean this one up completely.

Summary

- TLS is a secure protocol, few changes over the years
  - SSL is now depreciated
  - Version 1.3 have been introduced due to various attacks
- Security protocols have some common properties:
  - Negotiate algorithms and ciphers to use
  - Always derive new key material
  - Only use private keys for authentication – enforce perfect forward secrecy
  - Change keys regularly
- Random number generation and use of strong ciphers essential
- Attackers will not break the ciphers
  - Heartbleed: send message with incorrect headers
  - Logjam: MITM degrades ciphers to make it crackable