TLS 1.3

Evolutionary History and Innovations

ISSS Webinar, June 24, 2021

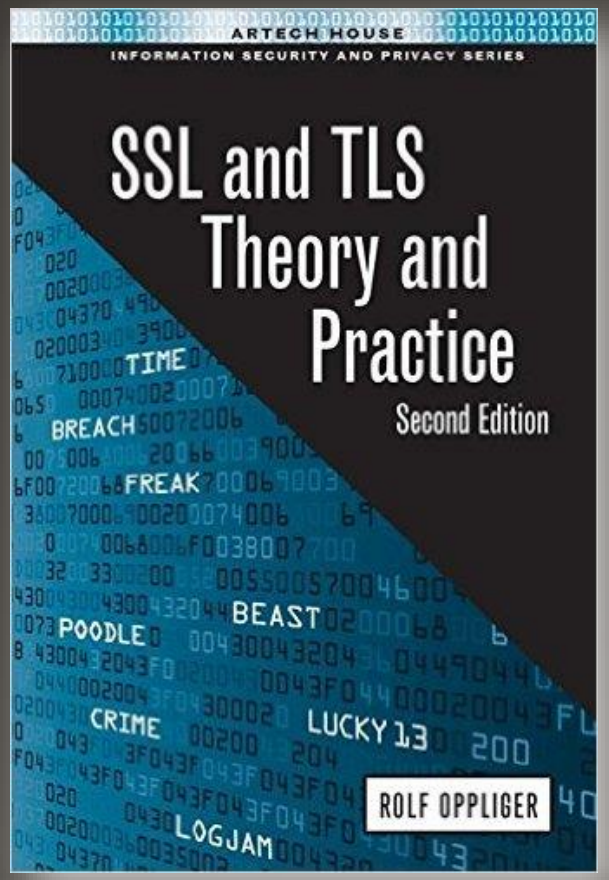
whoami

- eSECURITY Technologies Rolf Oppliger (founder and owner)
- Swiss National Cyber Security Centre NCSC (scientific employee)
- University of Zurich (adjunct professor)
- Artech House (author and series editor for information security and privacy)

→ rolf-oppliger.ch or rolf-oppliger.com

eSECURITY® Academy esecurity.academy

Reference Book



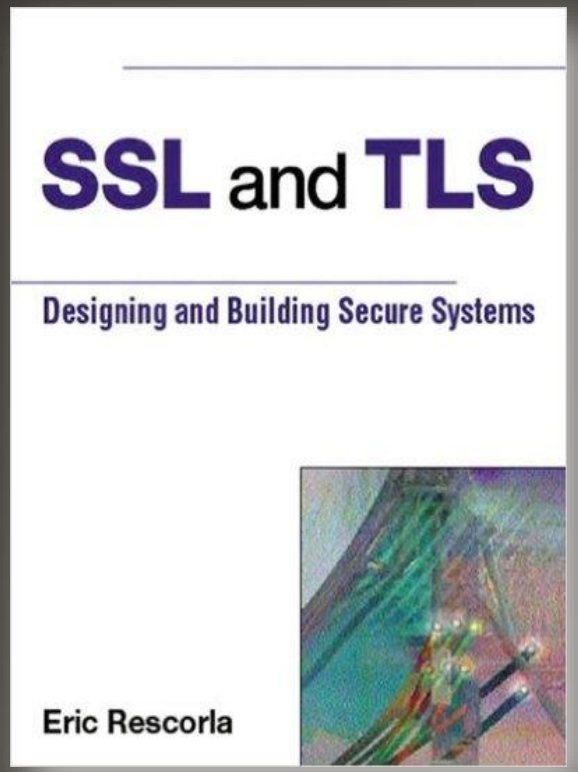
ISBN 978-1-60807-998-8

→ https://www.esecurity.ch/Books/ssltls2e.html

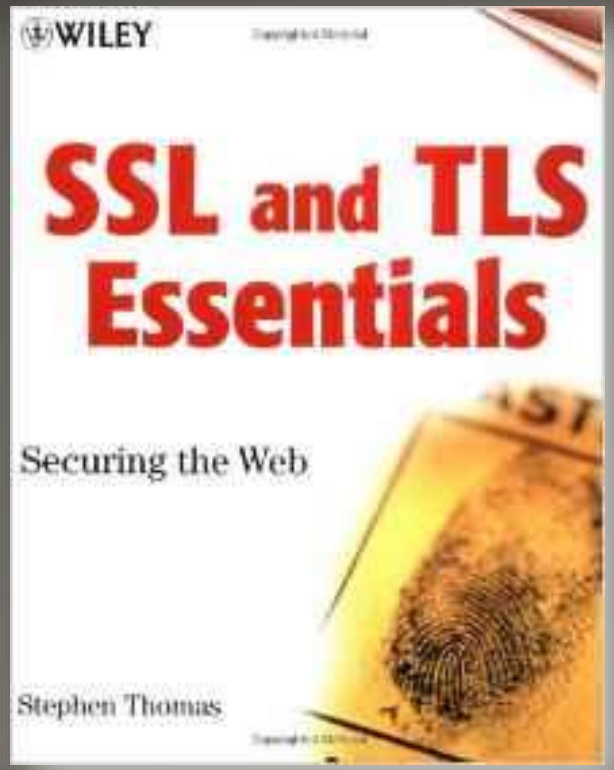
Web site comprises TLS 1.2 and TLS 1.3 (in «middlebox compatibility mode») transcripts captured with Wireshark

Statistics about the use and deplyoment of the SSL/TLS protocols are available from several sources on the Internet (e.g., Qualys' SSL Labs, EFF SSL Observatory, ICSI Cerificate Notary, ...)

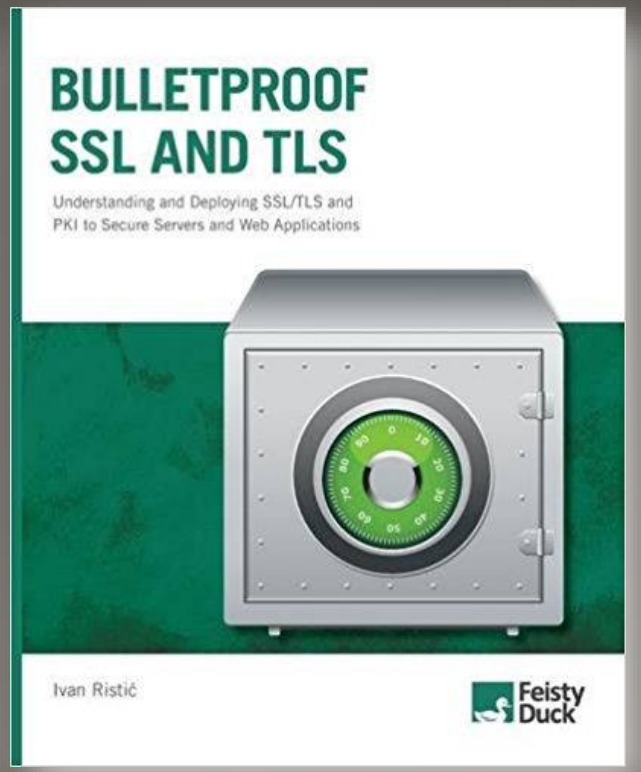
Complementary Reading



© 2000



© 2000



© 2014
(revised in 2017)

Outline

1. Introduction
2. Evolutionary History (SSL 3.0 … TLS 1.2)
3. Innovations (TLS 1.3)
4. Concluding Remarks

1. Introduction

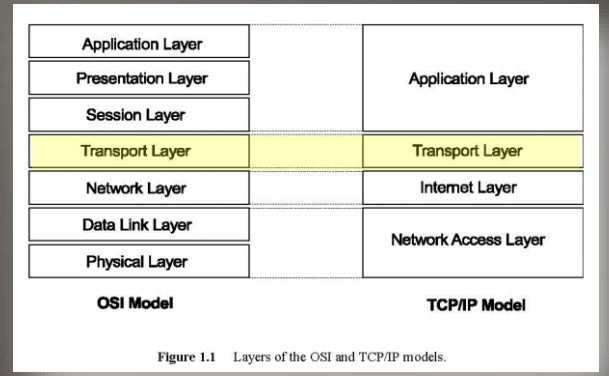
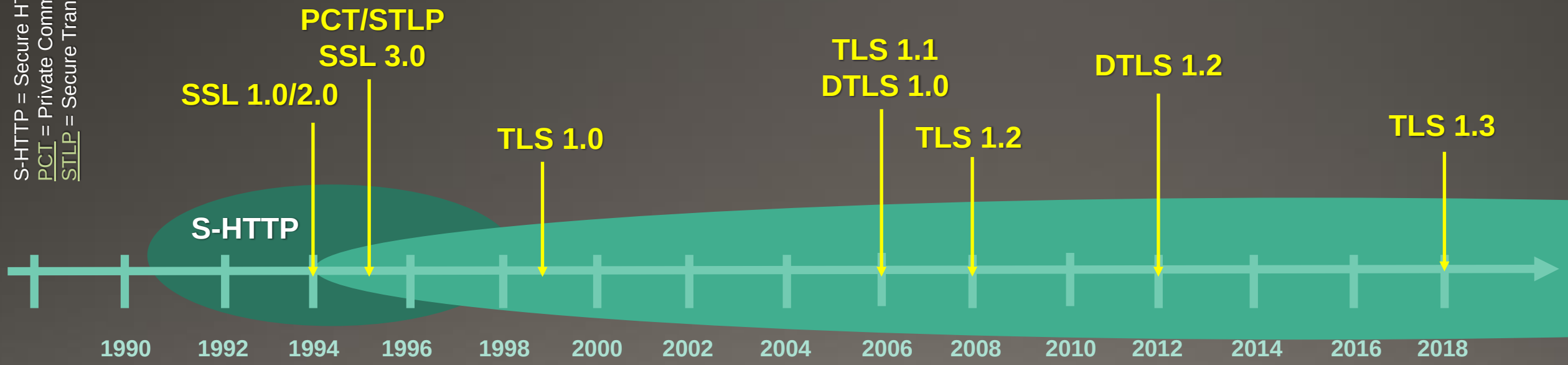


Figure 1.1 Layers of the OSI and TCP/IP models.

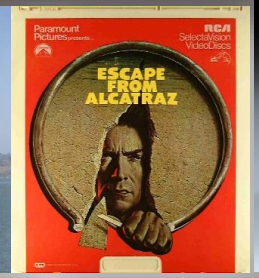
- There are many possibilities to use cryptographic techniques to implement security services
- Consequently, there are many cryptographic security protocols that work on different layers of the TCP/IP protocol stack (TCP/IP model)

S-HTTP = Secure HTTP (experimental RFC 2660)
PCT = Private Communication Technology
STLP = Secure Transport Layer Protocol



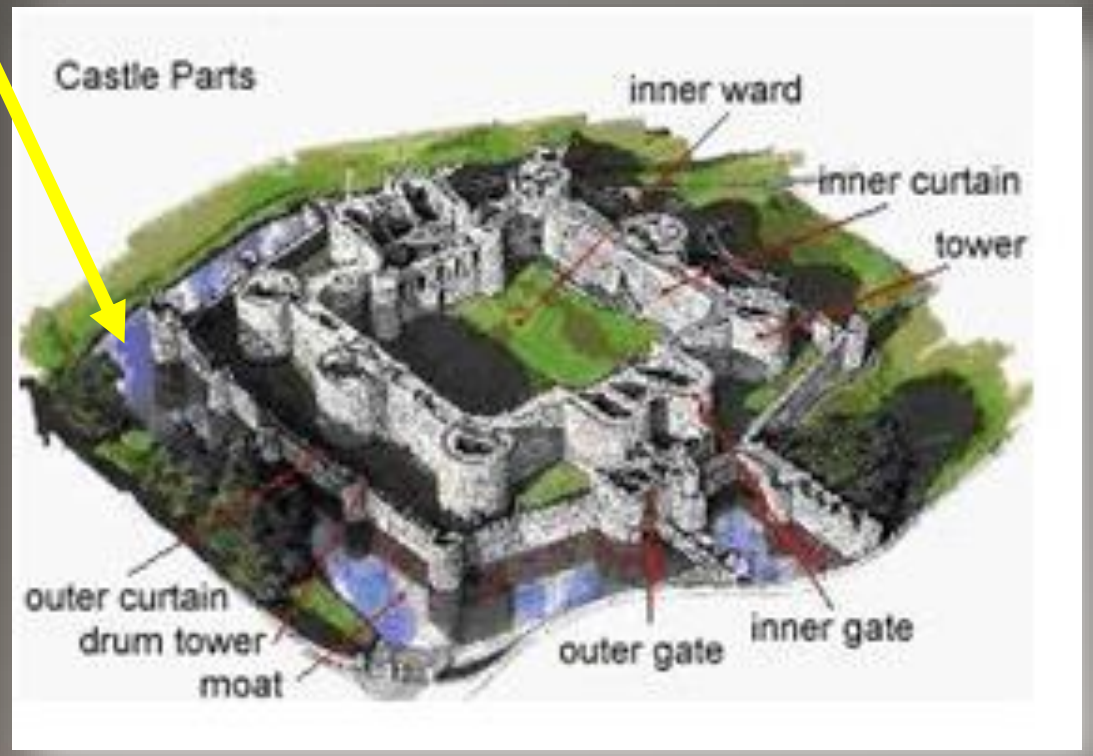
- The SSL/TLS (and DTLS) protocols are very widely deployed
- Main reasons
  - They can be used to secure any application-layer protocol
  - Their use is mostly transparent to the users
- The protocols have been designed to provide basic security services in the **Dolev-Yao model** (passive attacks)
- This model has limitations and shortcomings
- Most importantly, it does not take into account active attacks (e.g., mal-ware and attacks against the user interface)
- Today, many attacks are active or have at least an active component (→ some attacks may work in spite of the SSL/TLS protocols)

- An absolute notion of security can only be achieved in theory (i.e., in a «clean» and well-defined model)
- In practice, almost all implementations deviate from such a model
- Consequently, they have vulnerabilities and weaknesses that may be exploited (e.g., side-channel attacks)
- In such a situation, it is common to play **«cops and robbers» games**
  - «Secure» money transport
  - «Burglar-proof» safe
  - «Escape-proof» prison
  - ...



- This also applies to the SSL/TLS protocols

- A medieval castle may serve as an analogy
  - Sometimes it needs to be patched
  - Sometimes it needs to be protected with additional defenses (counter-measures)
- Both approaches are important (short-term vs. long-term)
- But they may also be subject to counterattacks
- This keeps the «cops and robbers» game up and running



2. Evolutionary History (SSL 3.0 … TLS 1.2)

- The SSL protocol was developed in the 1990s (→ historic RFC 6101)
- It provides the following **security services**
  - Peer entity authentication service
  - Data authentication service
  - Connection confidentiality service
  - Connection integrity service (without recovery)

NETSCAPE

- It uses the following **security mechanisms**
  - Encipherment
  - Digital signature mecha-nisms
  - Data integrity mechanisms
  - Authentication exchange mechanisms

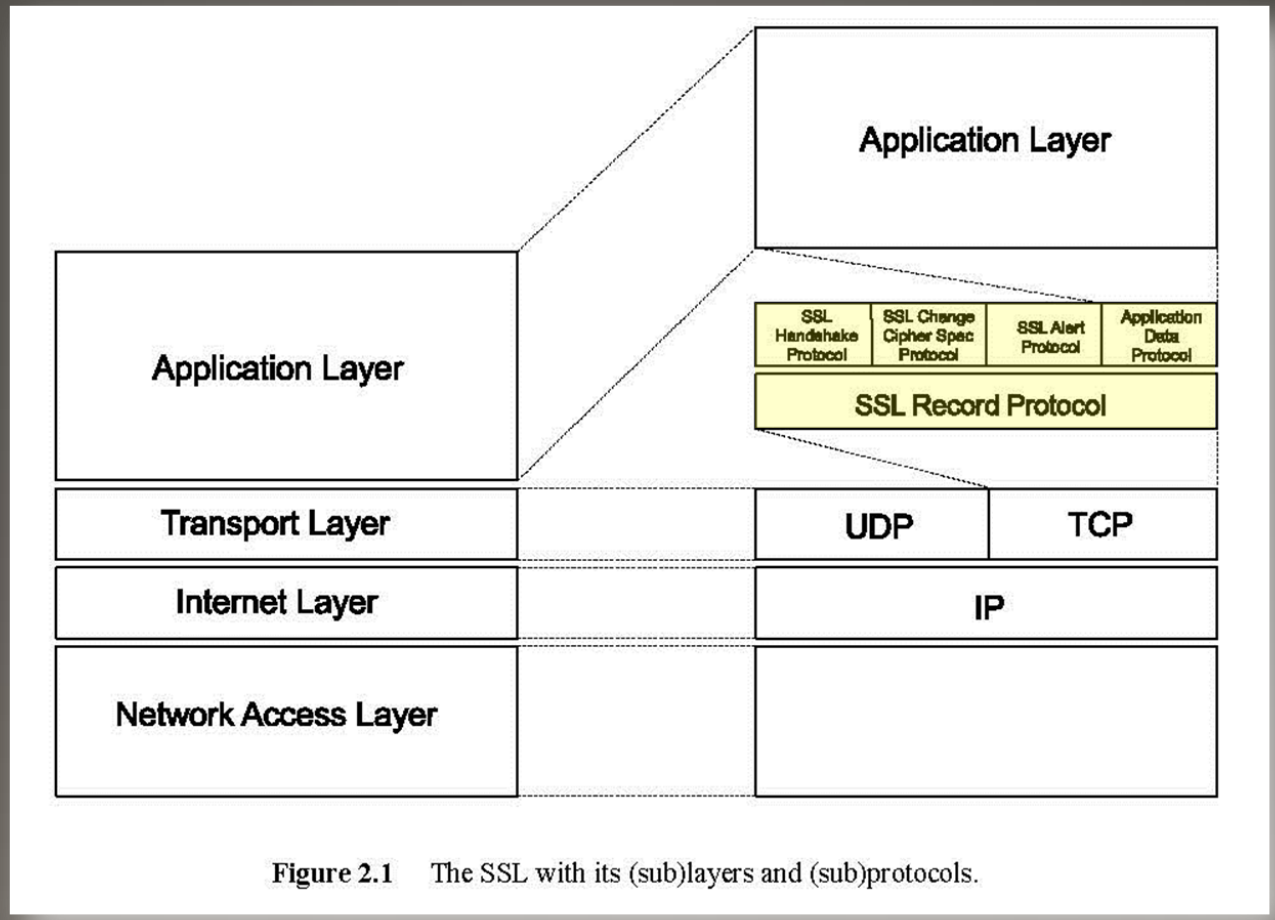


Figure 2.1 The SSL with its (sub)layers and (sub)protocols.

SSL Record Protocol

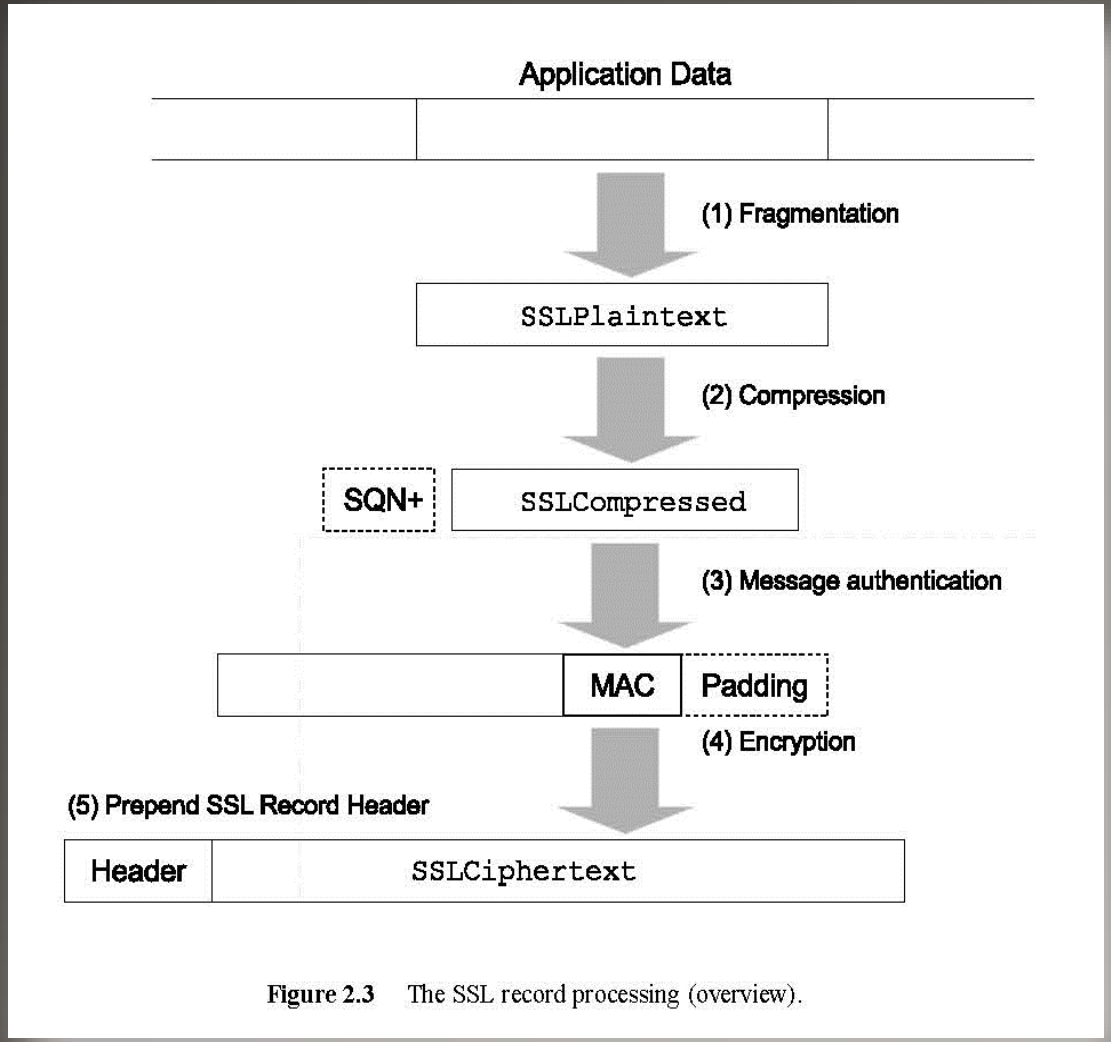


Figure 2.3 The SSL record processing (overview).

- Each SSL record consists of
  - Type (1 byte)
  - Version (2 bytes) `0x0300 = 3,0`
  - Length (2 bytes) $< 2^{14} = 16{,}384$
  - Fragment (variable length)

20 = Change Cipher Spec
21 = Alert
22 = Handshake
23 = Application Data

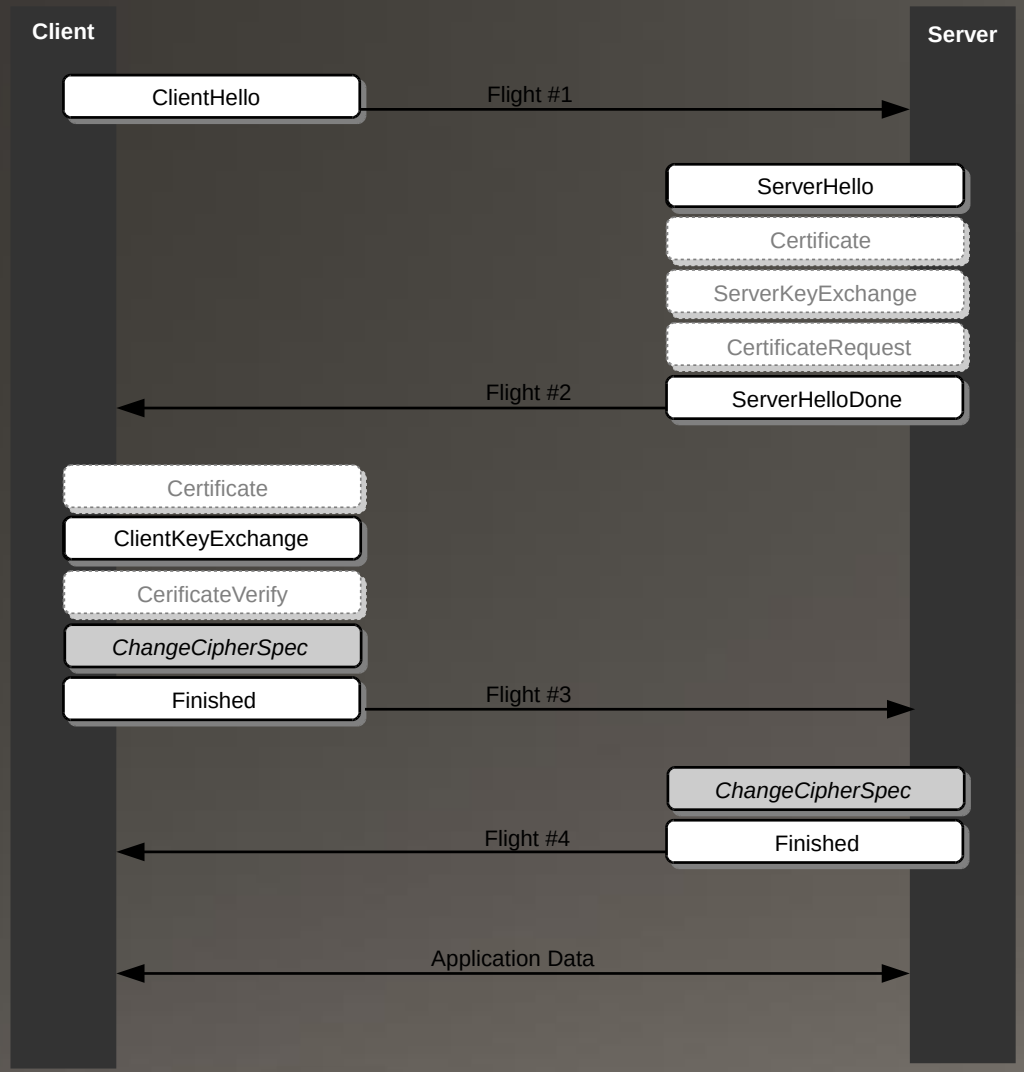
- The SSL record protocol follows the **Authenticate-then-Encrypt (AtE)** approach

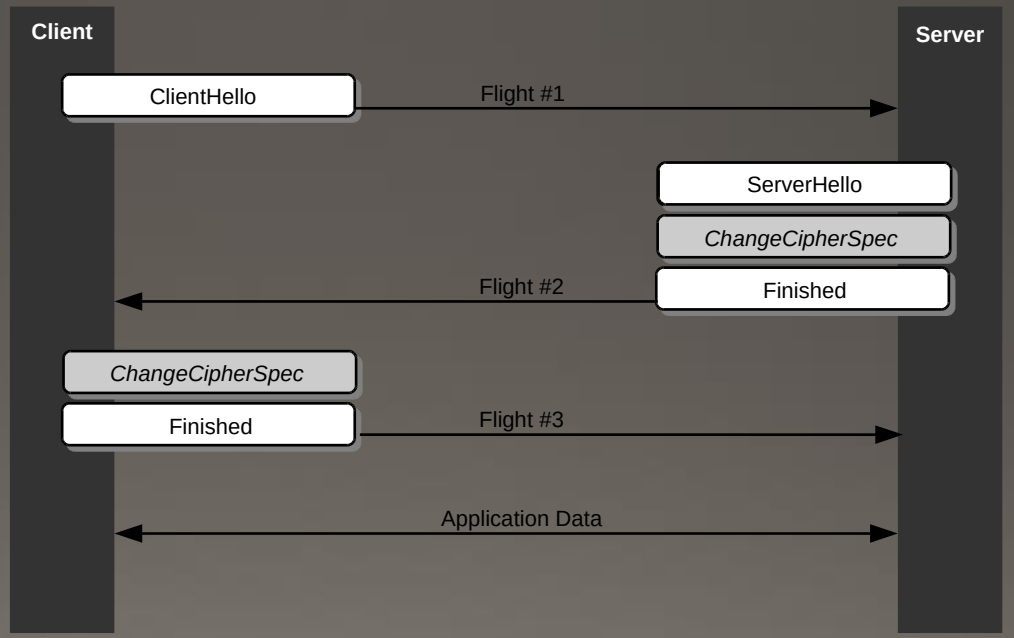
Table 2.4
SSL Cipher Suites (according to [1])

Name of the cipher suite	Key Exchange	Cipher	Hash
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA_EXPORT	RC4_40	MD5
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA_EXPORT	RC2_CBC_40	MD5
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA_EXPORT	DES40_CBC	SHA
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES_CBC	SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	DH_DSS_EXPORT	DES40_CBC	SHA
SSL_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	DH_RSA_EXPORT	DES40_CBC	SHA
SSL_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES_CBC	SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DHE_DSS_EXPORT	DES40_CBC	SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES_CBC	SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	DHE_RSA_EXPORT	DES40_CBC	SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES_CBC	SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	DH_anon_EXPORT	RC4_40	MD5
SSL_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH_anon	DES40_CBC	SHA
SSL_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES_CBC	SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
SSL_FORTEZZA_KEA_WITH_NULL_SHA	FORTEZZA_KEA	NULL	SHA
SSL_FORTEZZA_KEA_WITH_FORTEZZA_CBC_SHA	FORTEZZA_KEA	FORTEZZA_CBC	SHA
SSL_FORTEZZA_KEA_WITH_RC4_128_SHA	FORTEZZA_KEA	RC4_128	SHA

SSL_<key exchange>_WITH_<cipher + mode>_<hash>

SSL Handshake Protocol





- Multiple handshake messages (type 22) can be sent in a single SSL record

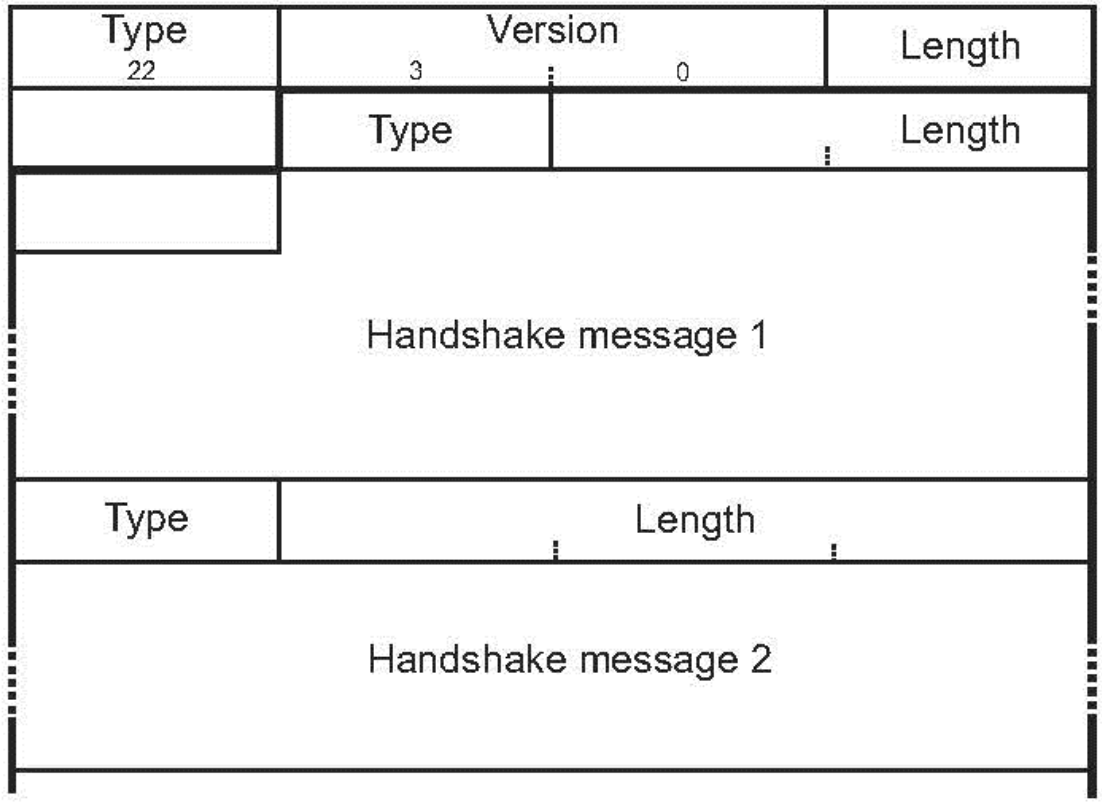


Figure 2.7 The general structure of an SSL handshake protocol message.

(messages from other sub-protocols cannot be packed into the same record)

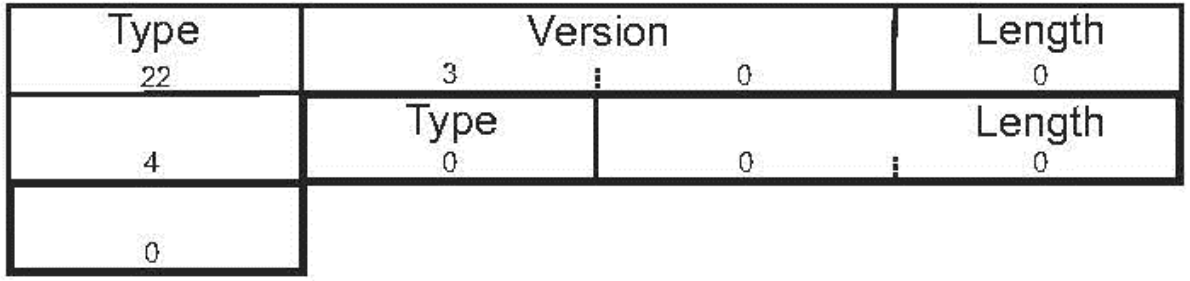


Figure 2.8 An SSL HELLOREQUEST message.

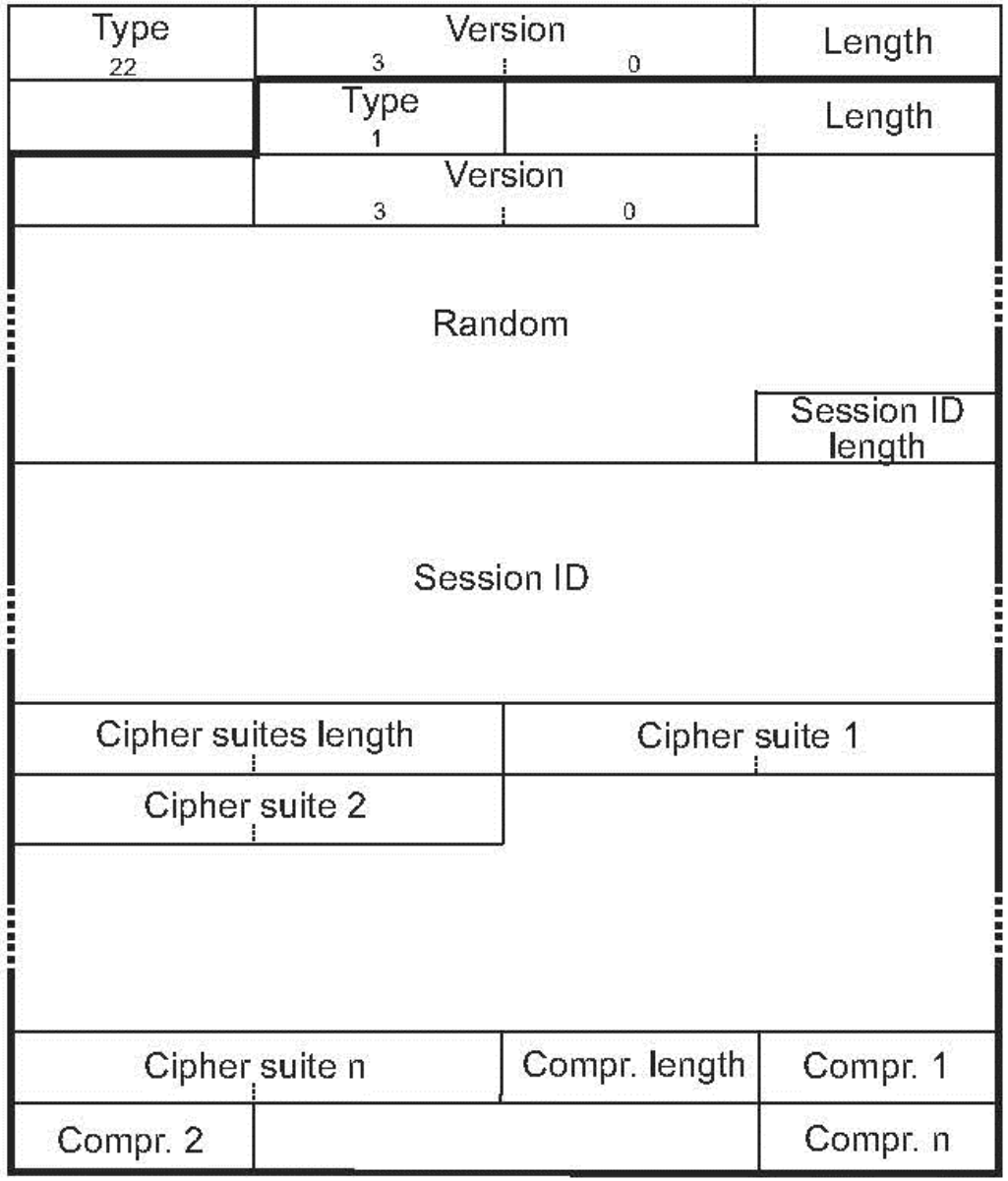


Figure 2.9 An SSL CLIENTHELLO message.

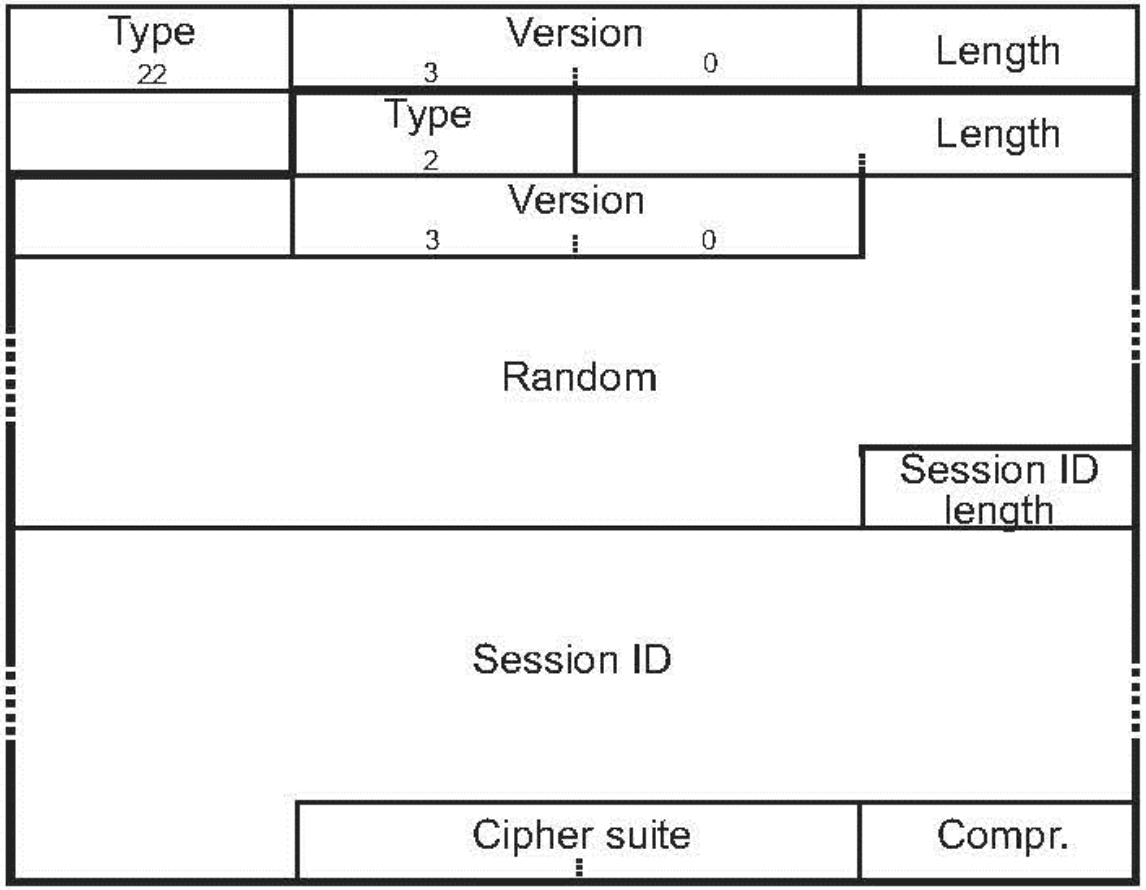


Figure 2.10 An SSL SERVERHELLO message.

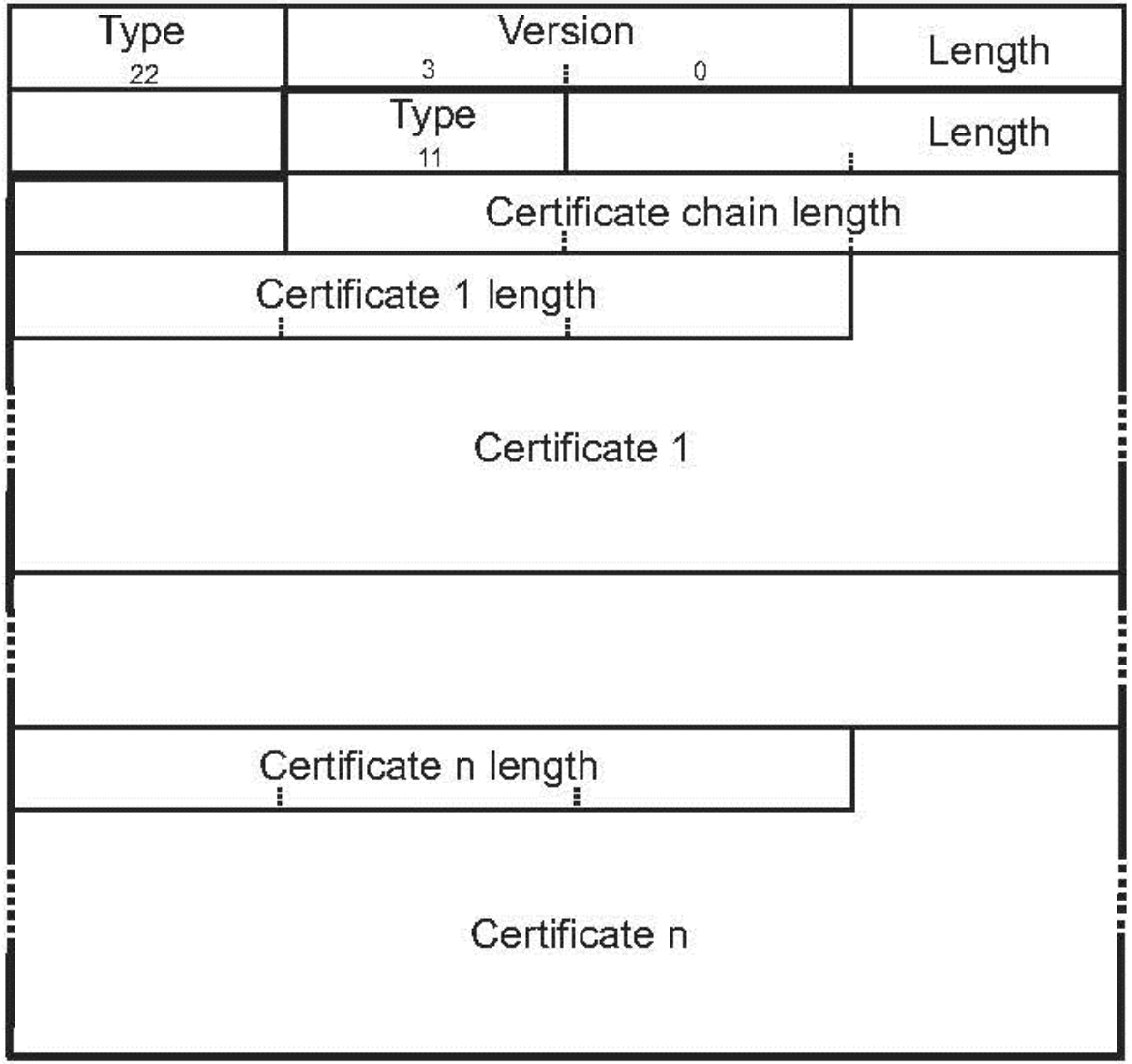


Figure 2.11 An SSL CERTIFICATE message.

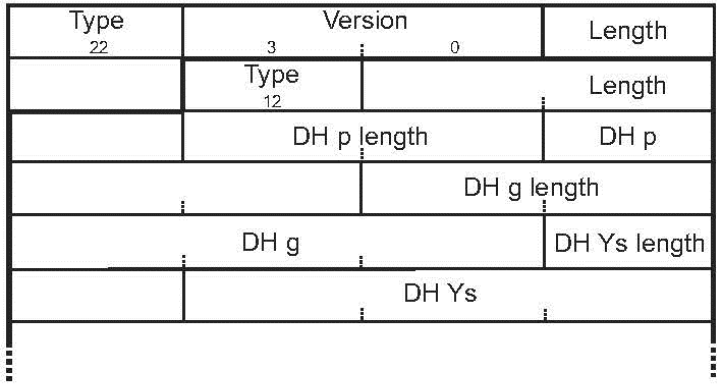


Figure 2.12 The beginning of an SSL SERVERKEYEXCHANGE message using Diffie-Hellman.

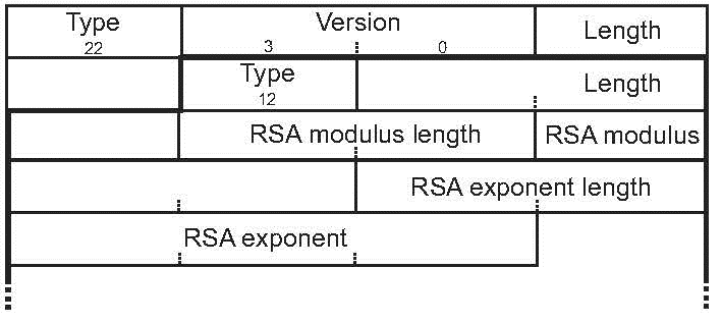


Figure 2.13 The beginning of an SSL SERVERKEYEXCHANGE message using RSA.

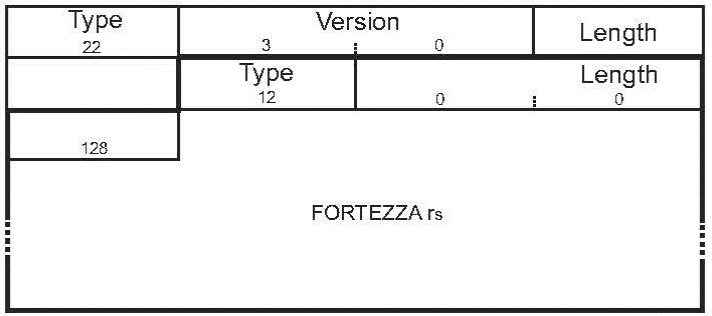


Figure 2.14 An SSL SERVERKEYEXCHANGE message using FORTEZZA.

Table 2.5
SSL Certificate Type Values

Value	Name	Description
1	rsa_sign	RSA signing and key exchange
2	dss_sign	DSA signing only
3	rsa_fixed_dh	RSA signing with fixed DH key exchange
4	dss_fixed_dh	DSA signing with fixed DH key exchange
5	rsa_ephemeral_dh	RSA signing with ephemeral DH key exchange
6	dss_ephemeral_dh	DSA signing with ephemeral DH key exchange
20	fortezza_kea	FORTEZZA signing and key exchange

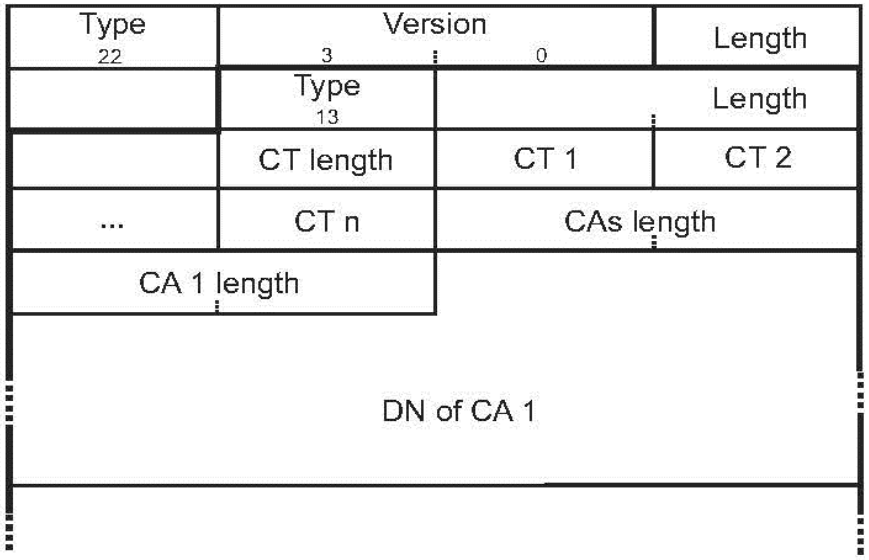


Figure 2.15 An SSL CERTIFICATEREQUEST message.

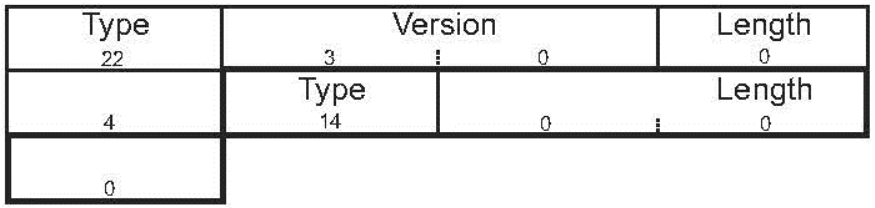


Figure 2.16 An SSL SERVERHELLODONE message.

CT =
certificate type

Table 2.6
FORTEZZA Key Material

Parameter	Size
Length of Y_C	2 bytes
Client's Y_C value for the KEA calculation	0–128 bytes
Client's R_C value for the KEA calculation	128 bytes
DSA signature for the client's KEA public key	40 bytes
Client's write key, wrapped by the TEK	12 bytes
Client's read key, wrapped by the TEK	12 bytes
IV for the client write key	24 bytes
IV for the server write key	24 bytes
IV for the TEK used to encrypt the premaster secret	24 bytes
Premaster secret, encrypted by the TEK	48 bytes

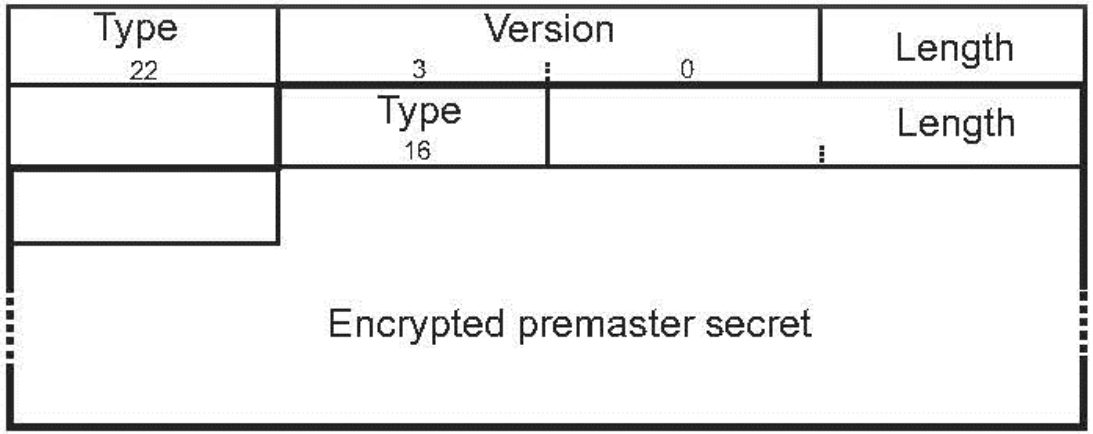


Figure 2.17 An SSL CLIENTKEYEXCHANGE message using RSA.

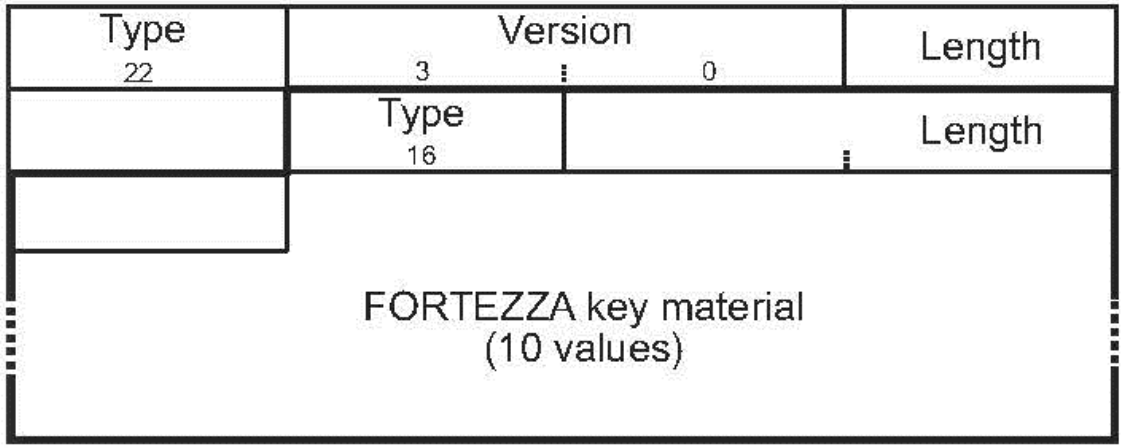


Figure 2.18 An SSL CLIENTKEYEXCHANGE message using FORTEZZA.

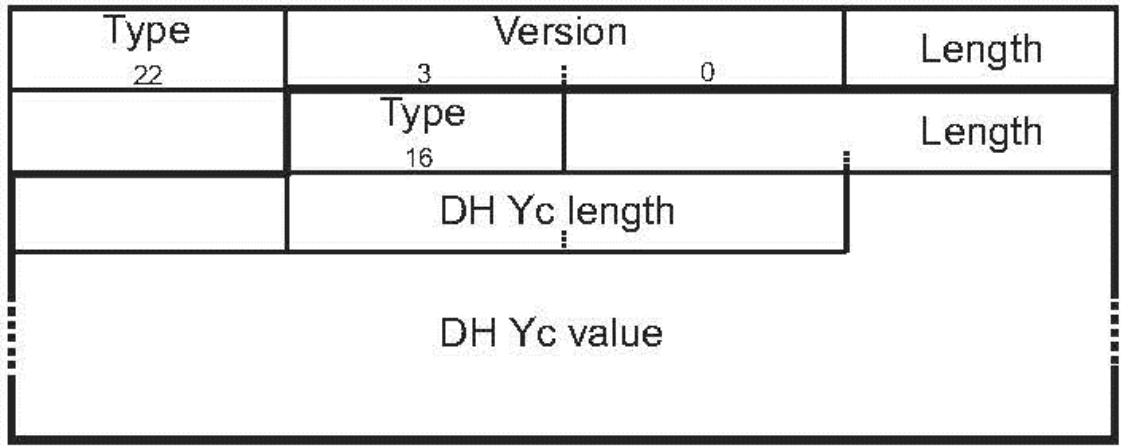


Figure 2.19 An SSL CLIENTKEYEXCHANGE message using Diffie-Hellman.

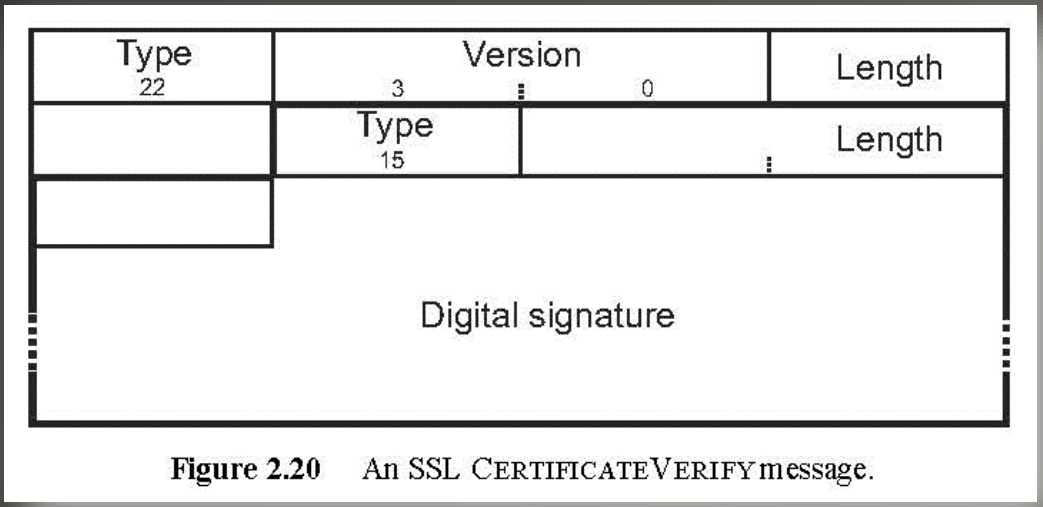


Figure 2.20 An SSL CERTIFICATEVERIFY message.

- RSA → Sign(MD5(H)‖SHA-1(H))
- DSA → Sign(SHA-1(H))

H

$$h(k \parallel opad \parallel h(handshake\_messages \parallel k \parallel ipad))$$

k = master secret

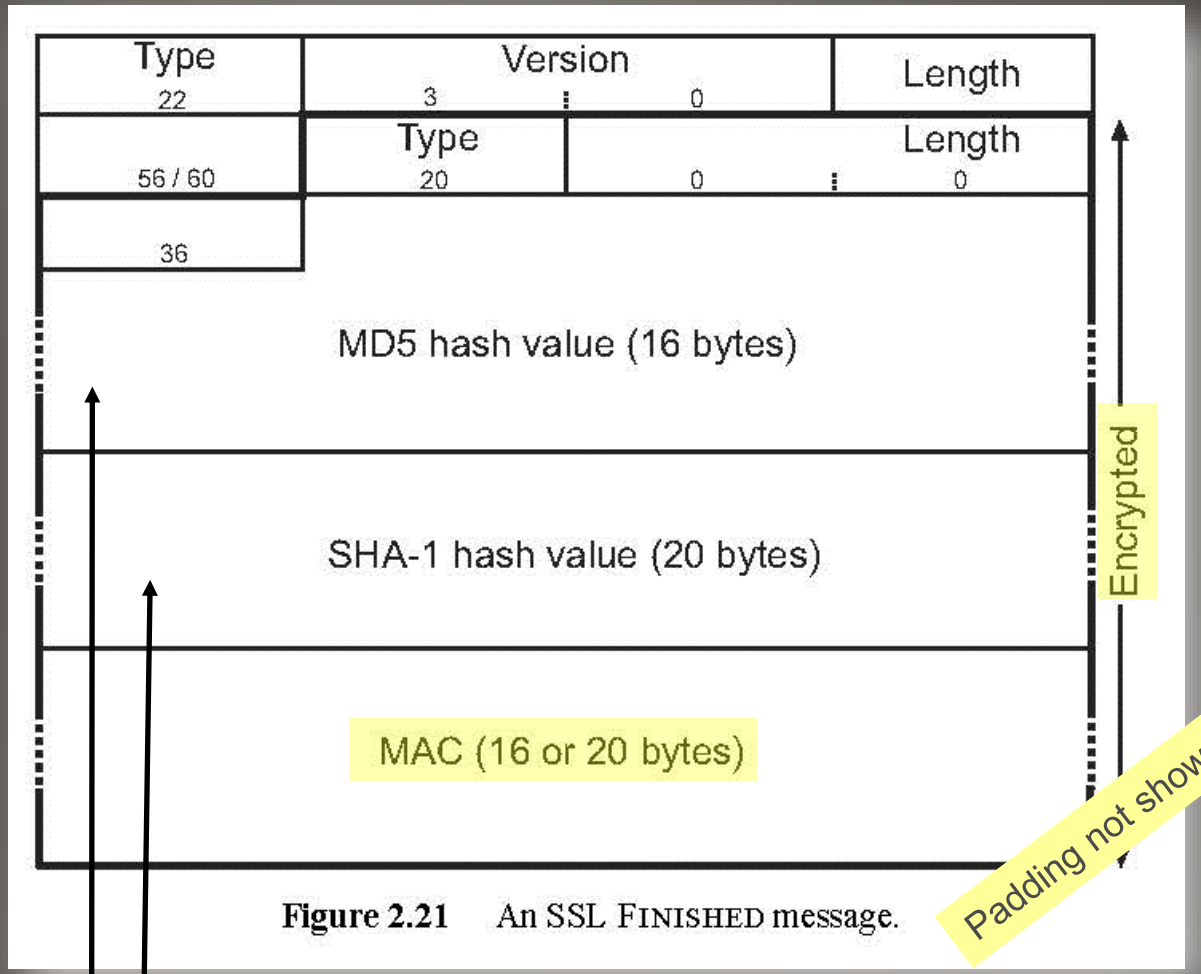


Figure 2.21 An SSL FINISHED message.

$$h(k \parallel opad \parallel h(handshake\_messages \parallel sender \parallel k \parallel ipad))$$

Client → 0x434C4E54

Server → 0x53525652

SSL Change Cipher Spec Protocol

- The SSL Change Cipher Spec Protocol consists of a single message (that is packed in a distinct record)

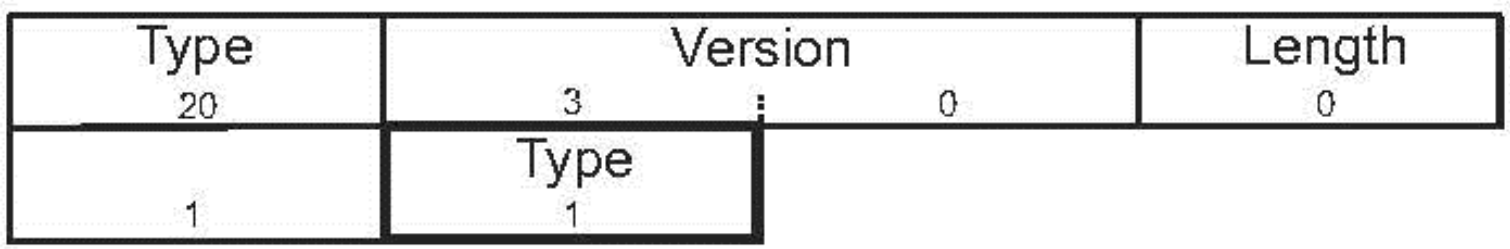


Figure 2.22 An SSL CHANGECIPHERSPEC message.

- The message triggers the SSL state machine
- The protocol is removed in TLS 1.3

SSL Alert Protocol

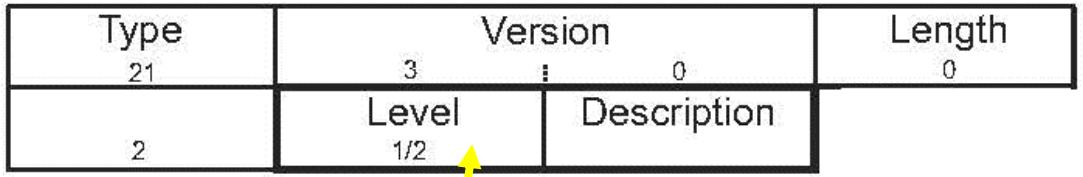


Figure 2.23 An SSL ALERT message.

1 → warning

2 → fatal

Table 2.7
SSL Alert Messages

Alert	Code	Brief description
close_notify	0	The sender notifies the recipient that it will not send any more messages on the connection. This alert is always a warning.
unexpected_message	10	The sender notifies the recipient that an inappropriate message was received. This alert is always fatal and should never be observed in communication between proper implementations.
bad_record_mac	20	The sender notifies the recipient that a record with an incorrect MAC was received. This alert is always fatal and should never be observed in communication between proper implementations.
decompression_failure	30	The sender notifies the recipient that the decompression function received improper input, meaning that it could not decompress the received data. This alert is always fatal and should never be observed in communication between proper implementations.
handshake_failure	40	The sender notifies the recipient that it was unable to negotiate an acceptable set of security parameters given the options available. This alert is always fatal.
no_certificate	41	The sender (which is always a client) notifies the recipient (which is always a server) that it has no certificate that can satisfy the server's certificate request. Note that this alert is only used in SSL (it is no longer used in TLS).
bad_certificate	42	The sender notifies the recipient that the certificate provided is corrupt (e.g., its contains a signature that cannot be verified correctly).
unsupported_certificate	43	The sender notifies the recipient that the certificate provided is of an unsupported type.
certificate_revoked	44	The sender notifies the recipient that the certificate provided has been revoked by the issuer.
certificate_expired	45	The sender notifies the recipient that the certificate provided has expired or is not currently valid.
certificate_unknown	46	The sender notifies the recipient that some unspecified issue arose in processing the certificate provided, rendering it unacceptable.
illegal_parameter	47	The sender notifies the recipient that a field in the SSL handshake message was out of range or inconsistent with some other field. This alert is always fatal.

No `decryption_failed` alert message in the SSL protocol

SSL Application Protocol

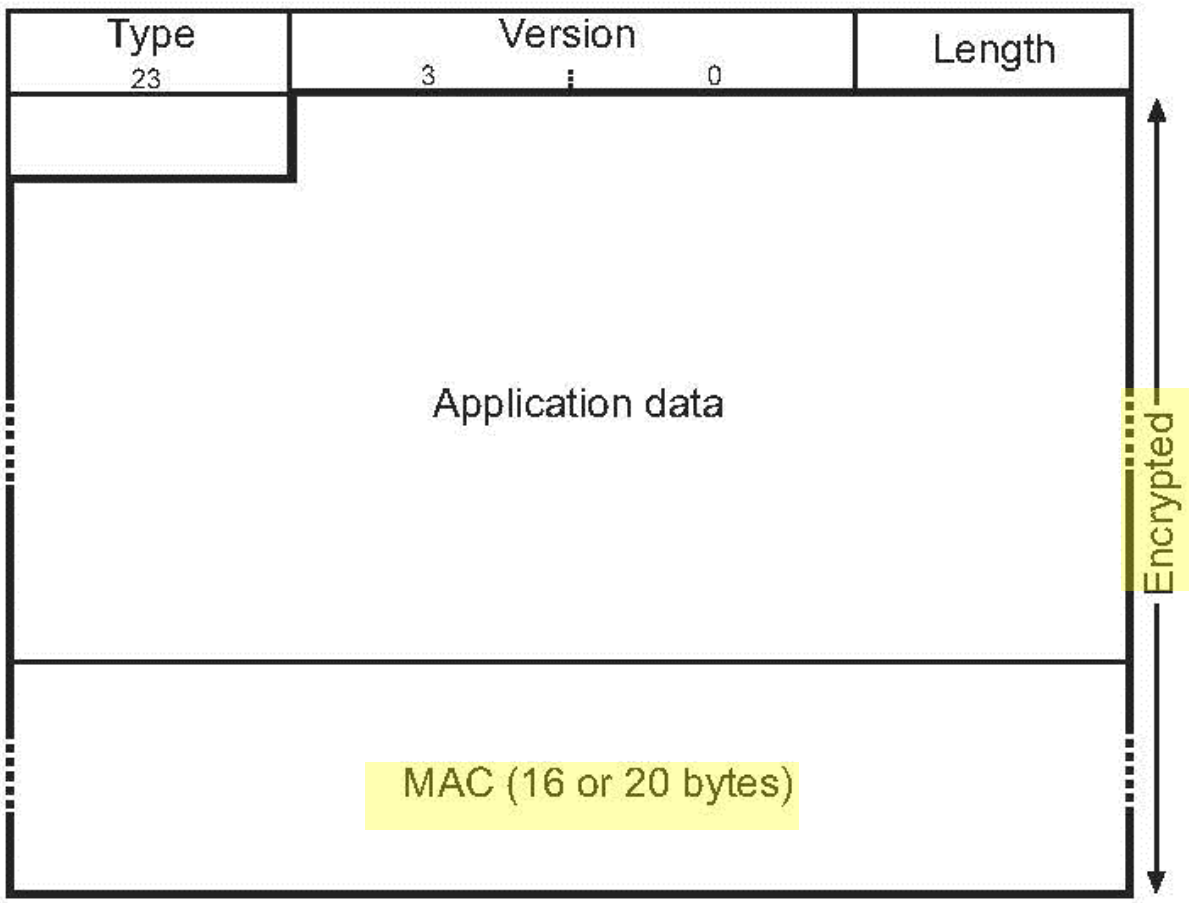


Figure 2.24 Application data encapsulated in an SSL record (stream cipher).

Padding not illustrated

- TLS 1.0 (1999) is similar to SSL 3.0
- Major changes
  - Fewer cipher suites (i.e., no FORTEZZA key exchange)
  - New security parameters and state elements for TLS connections
  - New TLS PRF to generate the keying material
  - HMAC construction that is now officially in line with RFC 2104
  - New alert messages (e.g., `decryption_failed` in addition to `bad_record_mac`)
  - Simplified CertificateVerify and Finished messages
  - …

$$HMAC_K(TLSCompressed) = h(K \parallel opad \parallel h(K \parallel ipad \parallel seq\_number \parallel \underbrace{type \parallel version \parallel length \parallel fragment}_{TLSCompressed}))$$

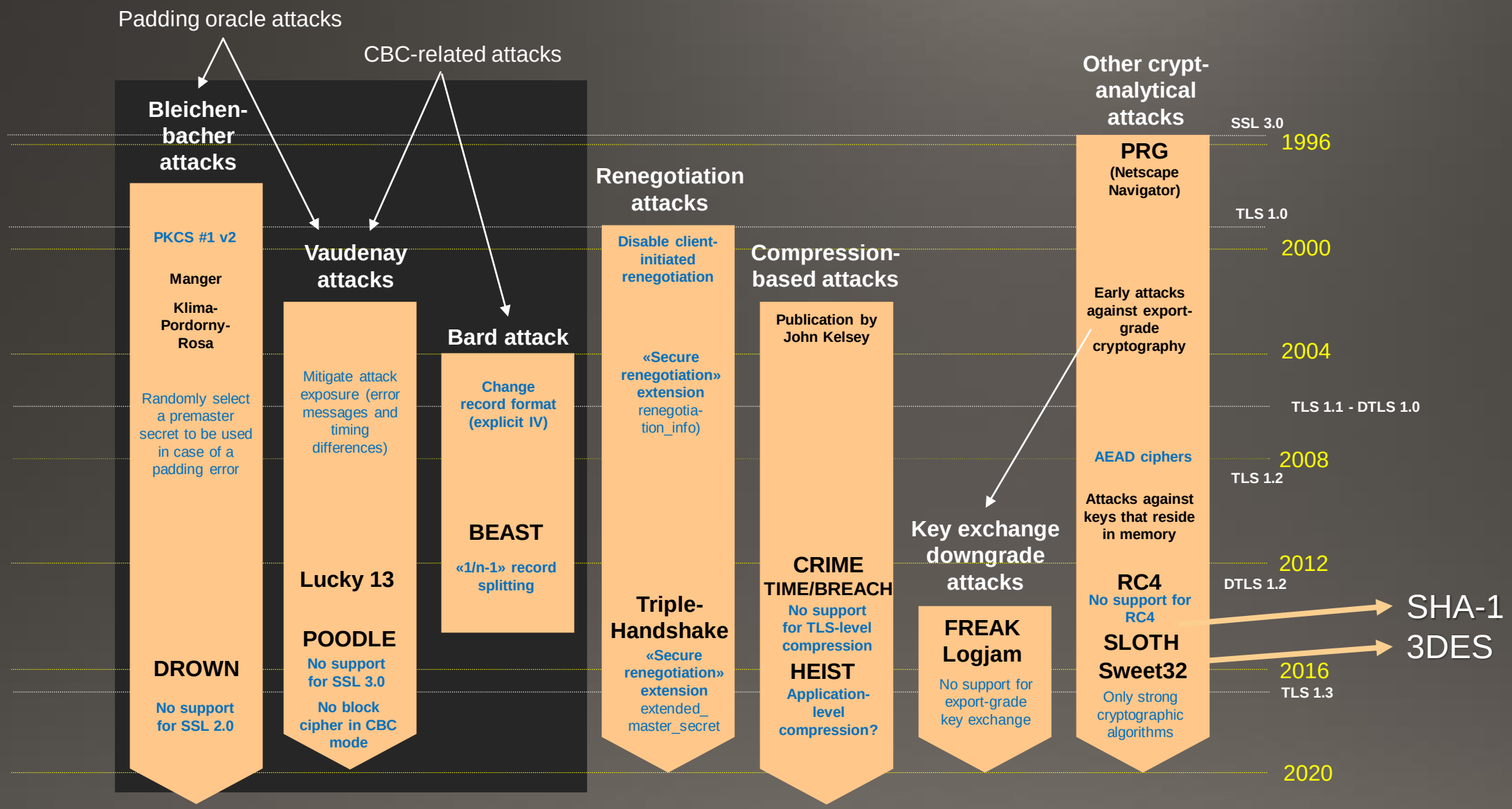
- TLS 1.1 (2006) is very similar to TLS 1.0
- Two major changes were motivated by cryptographic vulnerabilities that had been exploited by attacks against block ciphers operated in CBC mode
  - Serge Vaudeney's padding oracle attacks (2002/2003) → invalidate the `decryption_failed` alert message and always compute a MAC (even if the padding check fails) to destroy the timing channel [→ Lucky 13 attack]
  - Gregory Bard's publication on how to mount a blockwise CPA if the CBC IV is predictable (2004) [reused in the 2011 Browser Exploit Against SSL/TLS (BEAST) attack] → add an explcit IV to a TLS record
- TLS 1.1 also introduced a new way of specifying parameters and parameter values in IANA repositories (→ http://www.iana.org/assignments/tls-parameters)
- All other differences are subtle and not important

- TLS 1.2 (2008) is very similar to TLS 1.1
- The biggest change is the extension mechanism that allows additional functionality to be incorporated into TLS without having to change the underlying protocol (RFC 6066)
- The extensions are negotiated as part of the ClientHello and ServerHello messages (i.e., appended at the end of the messages)
- Each extension comprises a type field (2 bytes), a length field (2 bytes) and a variable length data field
- If a client wants to signal support for secure renegotiation, it appends 0xFF, 0x01, 0x00, 0x01, and 0x00 to the end of its ClientHello message
  - 0xFF01 refers to the type of the extension (= 65,281)
  - 0x0001 refers to the length of the data field (= 1)
  - 0x00 refers to the data field («empty»)

Table 3.11
TLS 1.2 Extension Types and Values

Extension Type	Value	Description	Reference
server_name	0	Server name indication	[15]
max_fragment_length	1	Maximum fragment length negotiation	[15]
client_certificate_url	2	Client certificate URL	[15]
trusted_ca_keys	3	Trusted CA keys	[15]
truncated_hmac	4	Truncated HMAC	[15]
status_request	5	Certificate status request	[15]
user_mapping	6	User mapping	[16]
client_authz	7	Client authorization	[17]
server_authz	8	Server authorization	[17]
cert_type	9	Certificate types	[18]
elliptic_curves	10	Elliptic curve cryptography	[19]
ec_point_formats	11	Elliptic curve cryptography	[19]
srp	12	SRP protocol	[20]
supported_signature_algorithms	13	Signature algorithms	[3]
use_srtp	14	Key establishment for the SRTP	[21]
heartbeat	15	Heartbeat	[22]
application_layer-protocol_negotiation	16	Application-layer protocol negotiation	[23]
status_request_v2	17	Certificate status request version 2	[24]
signed_certificate_timestamp	18	Certificate transparency	[25]
client_certificate_type	19	Raw public keys	[26]
server_certificate_type	20	Raw public keys	[26]
encrypt_then_mac	22	Use EtA instead of AtE	[27]
extended_master_secret	23	Secure renegotiation (revisited)	[28]
session_ticket	35	Session tickets	[29]
renegotiation_info	65281	Secure renegotiation	[30]

- Additional changes
  - Cipher suites that employ DES and IDEA are removed
  - Only 3DES and AES (block ciphers) or RC4 (stream cipher) prevail
  - Cipher suite `TLS_RSA_WITH_AES_128_CBC_SHA` is mandatory
  - Support for SHA-256, PSK, and ECC
  - Authenticated encryption with additional data (AEAD) → RFC 5116
    - Counter with CBC-MAC mode (CCM)
    - Galois/counter mode (GCM)
    - …
  - Compression algorithms NULL and DEFLATE (RFC 3749)
  - Exported keying material (EKM) according to RFC 5705 to mitigate MITM attacks



3. Innovations (TLS 1.3)

- Since the official release of TLS 1.2, the IETF TLS WG had been working on the next version of the TLS protocol
- The work was strongly influenced by the OPTLS protocol proposed by Hugo Krawczyk and Hoeteck Wee in 2015
- In August 2018, TLS 1.3 was specified in RFC 8446 (version 0x0304)
- The protocol comes along with improvements in terms of efficiency and security

- With regard to **efficiency**, the ultimate goal was to reduce the number of roundtrip times (RTTs) required to establish a session (because network latency is an increasingly important problem)
- To achieve **1-RTT**, the designers of TLS 1.3 adapted ideas from False Start and Snap Start (originally developed by Google)
- The client tries to opportunistically guess the key exchange method preferred by the server and provides its respective key share in the first flight (i.e., as part of the ClientHello message)
- If the client and server share a PSK, then the ClientHello message may comprise encrypted data («early data»), and hence provide **0-RTT**
- In this case, no certificate handling is required (because it is assumed that the entities have already authenticated themselves)

- 0-RTT has several security subtleties (and its use should therefore be considered with care)
  - It is susceptible to replay attacks

    → anti-replay mechanisms and applications must be idempotent (i.e., messages sent multiple times must not change the server state)
  - It is susceptible to DoS attacks

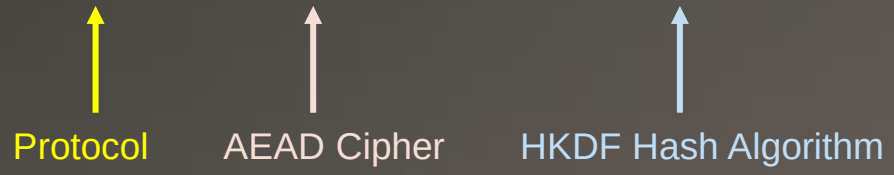
    → amount of «early data» must be limited to a reasonable size
  - The «early data» is not forward secure

    → application must be aware of this fact and take it into account
- This requires a lot of care taken by the application software developers
- The usefulness of 0-RTT is controversially discussed in the community

- With regard to **security**, TLS 1.3 is made as simple as possible (to make it less susceptible to implementation and configuration flaws)
- Also, it only uses cryptographic primitives and options that are known (or commonly believed) to be secure
  - No compression (other than NULL)
  - No session resumption or renegotiation (PSK-based key exchange instead of session IDs and session tickets)
  - Key exchange is always based on PSK, (EC)DHE, or a combination thereof (→ no static RSA or DH → controversial topic within the IETF and ETSI → Middlebox Security or Enterprise Transport Security (ETS) protocol)
  - Authentication is based on PSK, RSA (~ RSASSA-PSS), ECDSA, or EdDSA (e.g., Curve25519, Ed448-Goldilocks, …)
  - The TLS PRF is replaced with a HMAC-based Key Derivation Function (HKDF) that is in line with RFC 5869 (using SHA256 or SHA384)

- TLS 1.3 cipher suites only comprise an AEAD cipher (+ key length) and a hash function for the HKDF

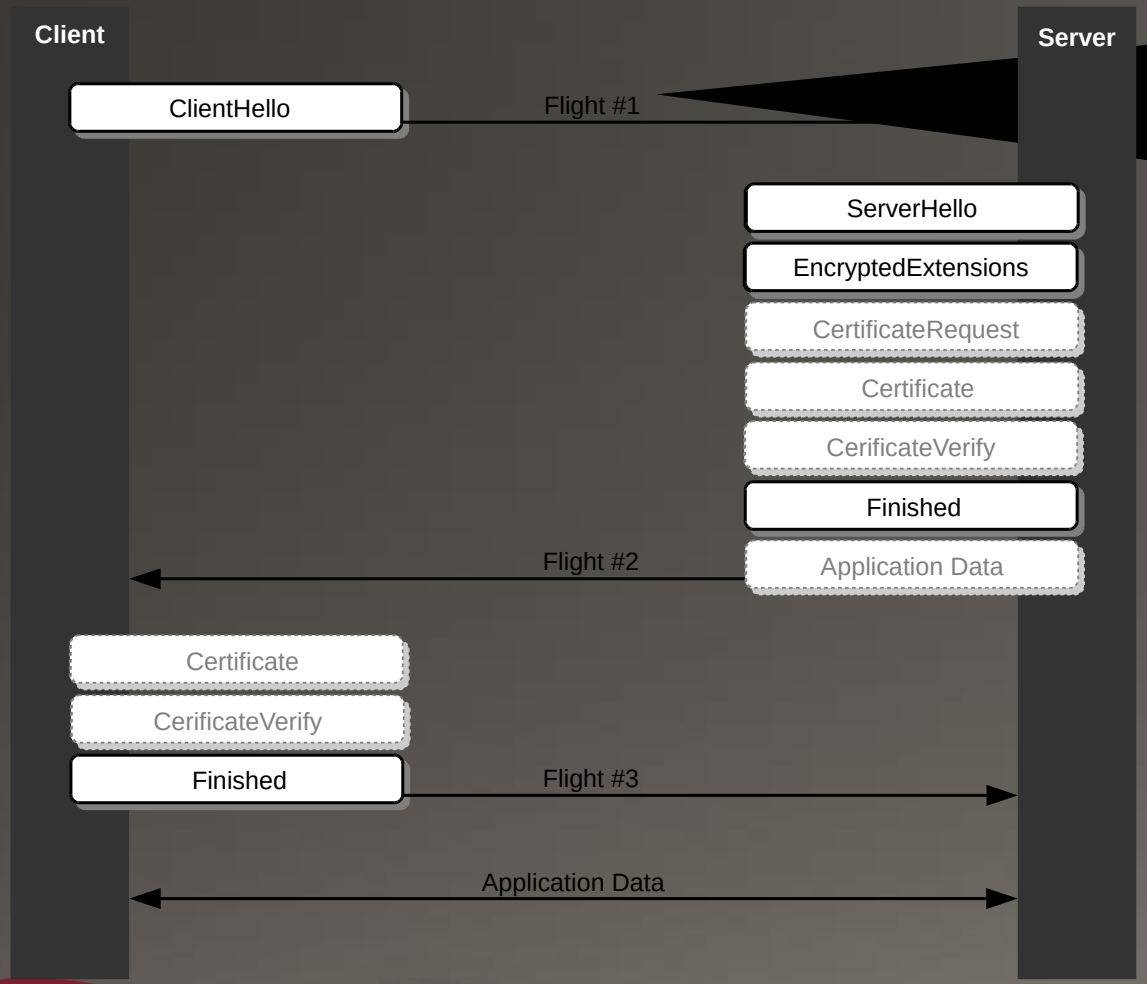
TLS_AES_128_GCM_SHA256	(0x13,0x01) → RFC 5116	**MUST implement**
TLS_AES_256_GCM_SHA384	(0x13,0x02)	**SHOULD implement**
TLS_CHACHA20_POLY1305_SHA256	(0x13,0x03) → RFC 7539	
TLS_AES_128_CCM_SHA256	(0x13,0x04)	
TLS_AES_256_CCM_8_SHA256	(0x13,0x05) → RFC 6655	



- The TLS 1.3 cipher suites are itemized in the same TLS parameters repository maintained by the IANA (first byte is 0x13)

- In TLS 1.3, all handshake messages after the ServerHello are encrypted
- This improves the confidentiality of the handshake
- It means, for example, that certificates are no longer sent in the clear
- This also makes it more important to send the SNI in encrypted form [→ Encrypted SNI (ESNI) / Encrypted ClientHello (ECH)]
- Since July 2020, the Great Firewall (GFW) of China has been blocking TLS 1.3 connections using ESNI
- More generally, all extensions that carry sensitive information should no longer be sent in hello messages (that are sent in the clear)
- Instead, they should be sent in a new EncryptedExtensions message (that is encrypted by default)

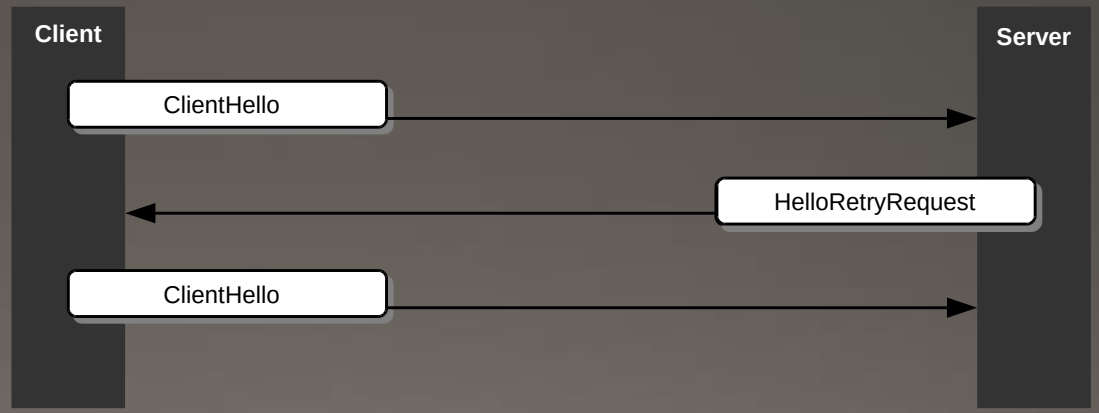
- TLS 1.3 message flow (simplified overview)



In addition to the usual extensions, this message may also include

- **`pre_shared_key`** → PSKs known to the client
- **`psk_key_exchange_modes`** → PSK alone or with (EC)DHE
- **`supported_groups`** → supported (EC)DHE groups
- **`key_share`** → (EC)DHE key shares for some or all of the supported groups
- **`signature_algorithms`** → RSA, ECDSA, and/or EdDSA

If a PSK is used, then some «early data» can be encrypted and included in the ClientHello message (0-RTT)



Available in TLS 1.3

Table 3.11

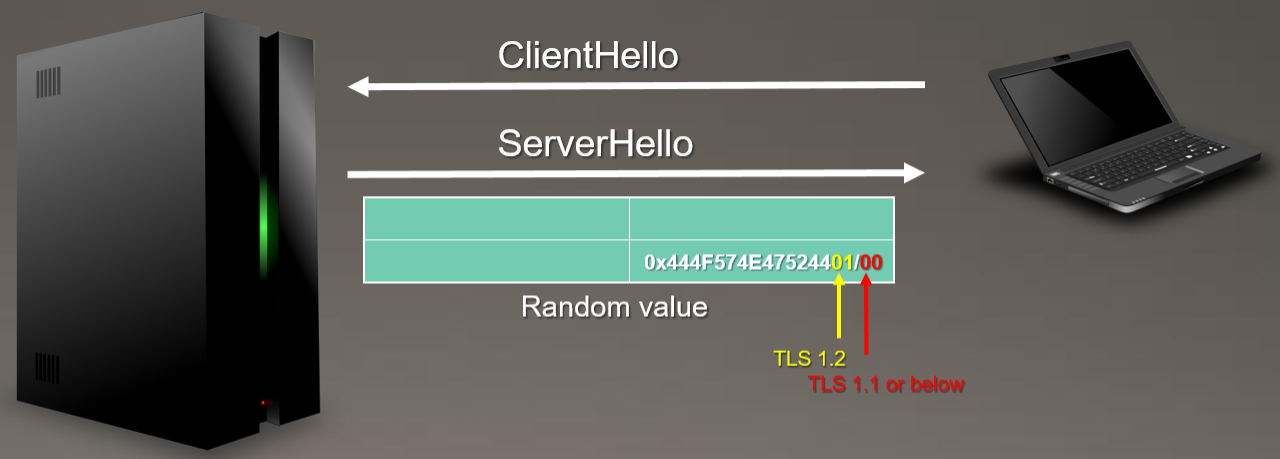
TLS 1.2 Extension Types and Values

Extension Type	Value	Description	Reference
server_name	0	Server name indication	[15]
max_fragment_length	1	Maximum fragment length negotiation	[15]
client_certificate_url	2	Client certificate URL	[15]
trusted_ca_keys	3	Trusted CA keys	[15]
truncated_hmac	4	Truncated HMAC	[15]
status_request	5	Certificate status request	[15]
user_mapping	6	User mapping	[16]
client_authz	7	Client authorization	[17]
server_authz	8	Server authorization	[17]
cert_type	9	Certificate types	[18]
elliptic_curves → «supported_groups»	10	Elliptic curve cryptography	[19]
ec_point_formats	11	Elliptic curve cryptography	[19]
srp	12	SRP protocol	[20]
~~supported~~_signature_algorithms	13	Signature algorithms	[3]
use_srtp	14	Key establishment for the SRTP	[21]
heartbeat	15	Heartbeat	[22]
application_layer-protocol_negotiation	16	Application-layer protocol negotiation	[23]
status_request_v2	17	Certificate status request version 2	[24]
signed_certificate_timestamp	18	Certificate transparency	[25]
client_certificate_type	19	Raw public keys	[26]
server_certificate_type	20	Raw public keys	[26]
encrypt_then_mac	22	Use EtA instead of AtE	[27]
extended_master_secret	23	Secure renegotiation (revisited)	[28]
session_ticket	35	Session tickets	[29]
renegotiation_info	65281	Secure renegotiation	[30]

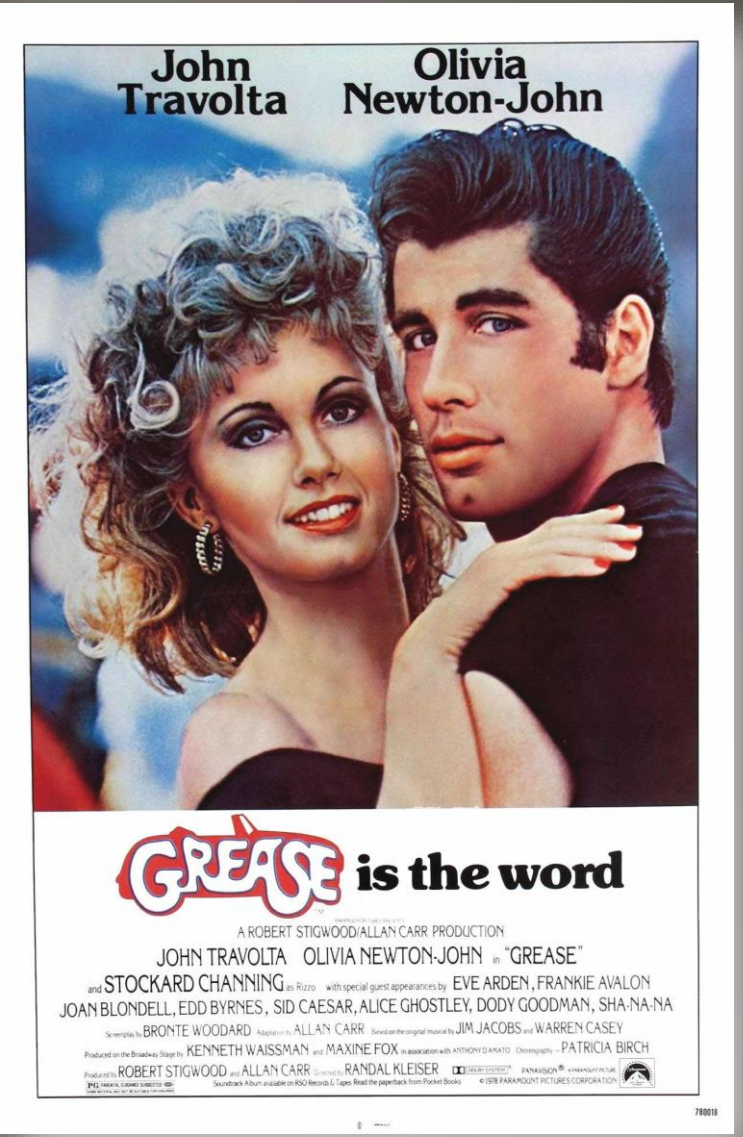
- Some new extensions in TLS 1.3 (→ IANA)
  - `supported_groups` (10) → RFC 4492, RFC 7919 → groups for (EC)DHE
  - `padding` (21) → RFC 7685 → to overcome bugs related to the message length
  - `token_binding` (24) → RFC 8472 → token binding according to RFC 8471
  - `compress_certificate` (27) → RFC 8879
  - `record_size_limit` (28) → RFC 8449 → improves `max_fragment_length`
  - `pwd_protect` (29), `pwd_clear` (30), and `password_salt` (31) → informational RFC 8492 specifying secure password suites for TLS
  - `ticket_pinning` (32) → experimental RFC 8672
  - `tls_cert_with_extern_psk` (33) → experimental RFC 8773
  - `delegated_credentials` (34) → Internet-Draft (front-end server can digitally sign on behalf of back-end server in a CDN)
  - `supported_ekt_ciphers` (39) → RFC 8870 → complements `use_srtp`

- `pre_shared_key` (41)
- `early_data` (42)
- `supported_versions` (43)
- `cookie` (44) → anti-clogging mechanism from IPsec and DTLS
- `psk_key_exchange_modes` (45) → PSK with/without (EC)DHE
- `certificate_authorities` (47) → replaces `trusted_ca_keys` extension
- `oid_filters` (48) → certificate extension OIDs (e.g., 2.5.29.15 for Key Usage)
- `post_handshake_auth` (49) → client signals support for post-handshake auth.
- `signature_algorithms_cert` (50) → complements the `signature_algorithms` extension for certificates
- `key_share` (51) → (EC)DHE parameter
- `transparency_info` (52) → Internet-Draft
- …

- TLS 1.3 version negotaion is different than in previous versions
  - The ClientHello and ServerHello messages have a `legacy_version` field that is constantly set to 0x0303 (TLS 1.2)
  - In addition, there is a `supported_versions` extension (43) that comprises the value 0x0304 (TLS 1.3)
- Also, in previous versions of the SSL/TLS protocols, the TLS_FALL-BACK_ SCSV was used to protect against some protocol downgrade attacks (mainly to mitigate the POODLE attack)
- TLS 1.3 provides another (simple) anti-downgrade mechanism
  - Client must check that neither of the two values occurs in the random value of the server (otherwise something fishy is going on)



- The version negotiation and extensibility mechanisms of TLS have often led to protocol failures in the past
- In January 2020, a countermechanism called Generate Random Extensions And Sustain Extensibility (GREASE) was proposed in informational RFC 8701



4. Concluding Remarks

- SSL/TLS started as a relatively simple cryptographic security protocol
- With its modifications and extensions up to version 1.2, it has become an highly involved protocol with many features
- It supports all cryptographic technologies and techniques that are state-of-the art (e.g., AES, AEAD, ECC, HMAC, SHA-2, …)
- Whenever a new cryptographic primitive pops up, somebody writes an Internet-Draft or RFC that specifies its use in a TLS setting (e.g., PQC, quantum cryptography, …)
- The RFCs are experimental, informational, or submitted to the Internet standards track

- In addition to the ones addressed so far, there are many other vulnera-bilities and respective attacks reported in the media
  - Heartbleed
  - Apple's «goto fail»
  - Superfish
  - …

- Not all of attacks can be mitigated with cryptographic techniques
- Nevertheless, TLS 1.3 is a milestone in the evolution of a cryptographic security protocol for the transport layer
- The are only a few alternatives (e.g., Noise protocol framework)
- The (security) story will continue …

- Sources of recommendations and best practices
  - TLS Recommendations of the IETF (RFC 7525, BCP 195)
  - Security/Server Side TLS recommended configurations of the Mozilla Project
  - BSI TR-02102-2 «Cryptographic Mechanisms: Recommendations and Key Lengths: Use of Transport Layer Security (TLS)" Version: 2021-1»
  - SSL/TLS Deployment Best Practices of Qualys' SSL Lab
  - Transport Layer Protection Cheat Sheet of the Open Web Application Security Project (OWASP)
  - Revised guide about the use of TLS of the U.S. NIST
  - Security controls guidelines for SSL/TLS management of the SANS Institute
  - …

- Key recommendations from RFC 7525
  - Don’t support SSL (v2 or v3) anymore
  - Support TLS 1.0 or 1.1 only if needed
  - Support TLS 1.2 (TLS 1.3 if possible)
  - Disable SSL/TLS compression, unless the application takes care of the respective vulnerabilities
  - Don’t truncate HMAC values
  - Always provide forward secrecy (→ no static RSA or Diffie-Hellman key exchange)
  - Enforce the use of the following cipher suites
    - `TLS_DHE_RSA_WITH_AES_128_GCM_SHA256`
    - `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
    - `TLS_DHE_RSA_WITH_AES_256_GCM_SHA384`
    - `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`


thank you
gracias
danke
merci
obrigado
teşekkür ederim
go raibh maith agat
謝謝
ngiyabonga
bedankt
dziękuję
sukriya
kop khun krap
terima kasih
mochchakkeram
tapadh leat
спасибо
dank je
grazie
arigatô
dakujem
감사합니다
мерси
рахмат
Баярлалаа
faafetai lava
mersi
kia ora
barka
welalin
tack
spas
mahalo
paldies
grazzi
vinaka
blagodaram
misaotra
matondo
спасибі
kiitos
dankie
nanni
nandri
dhanyavad
maururu
köszönöm
hvala
bayarlalaa
gràçie
enkosi
хвала
asante
manana
obrigada
murakoze
tenki
djiere dieuf
tau
дякую
mamnun
chokrane
chnorakaloutioun
gratias ago
gràcies
sulpáy
akun
dankon
aćiū
sobodi
děkuji
sagolun
taiku
takk
trugarez
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
rahmet
kam sah hamnida
rahmat
তোমাকে ধন্যবাদ
xièxie
ευχαριστώ
diolch
dhanyavadagalu
shukriya
mercé