TLS/SSL

SSL/TLS

- Secure Sockets Layer (SSL) - deprecated
- Transport Layer Security (TLS)
- TLS 1.3 is faster and more secure
  - Less messages exchanged
  - Replaced RC4 with chacha20
  - Removed MD5

Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011 (RFC 6176)
SSL 3.0	1996	Deprecated in 2015 (RFC 7568)
TLS 1.0	1999	Deprecated in 2020[11][12][13]
TLS 1.1	2006	Deprecated in 2020[11][12][13]
TLS 1.2	2008	
TLS 1.3	2018	

SSL/TLS

- SSL/TLS is not used only by browsers and web servers…

- Linux/Microsoft for updates
- Ftp clients/servers
- Android applications
- Kodi
- Email applications
- …

There are many SSL/TLS libraries

- Google → BoringSSL written in C/C++, Go, Assembly
- Network Security Services → Mozilla, (used byFirefox) written in C, Assembly
- OpenSSL →C, Assembly used in Apache
- Secure Transport → Apple, used by Safari
- Bouncy Castle → Used in Android, written in Java,C#

TCP/IP and encapsulation

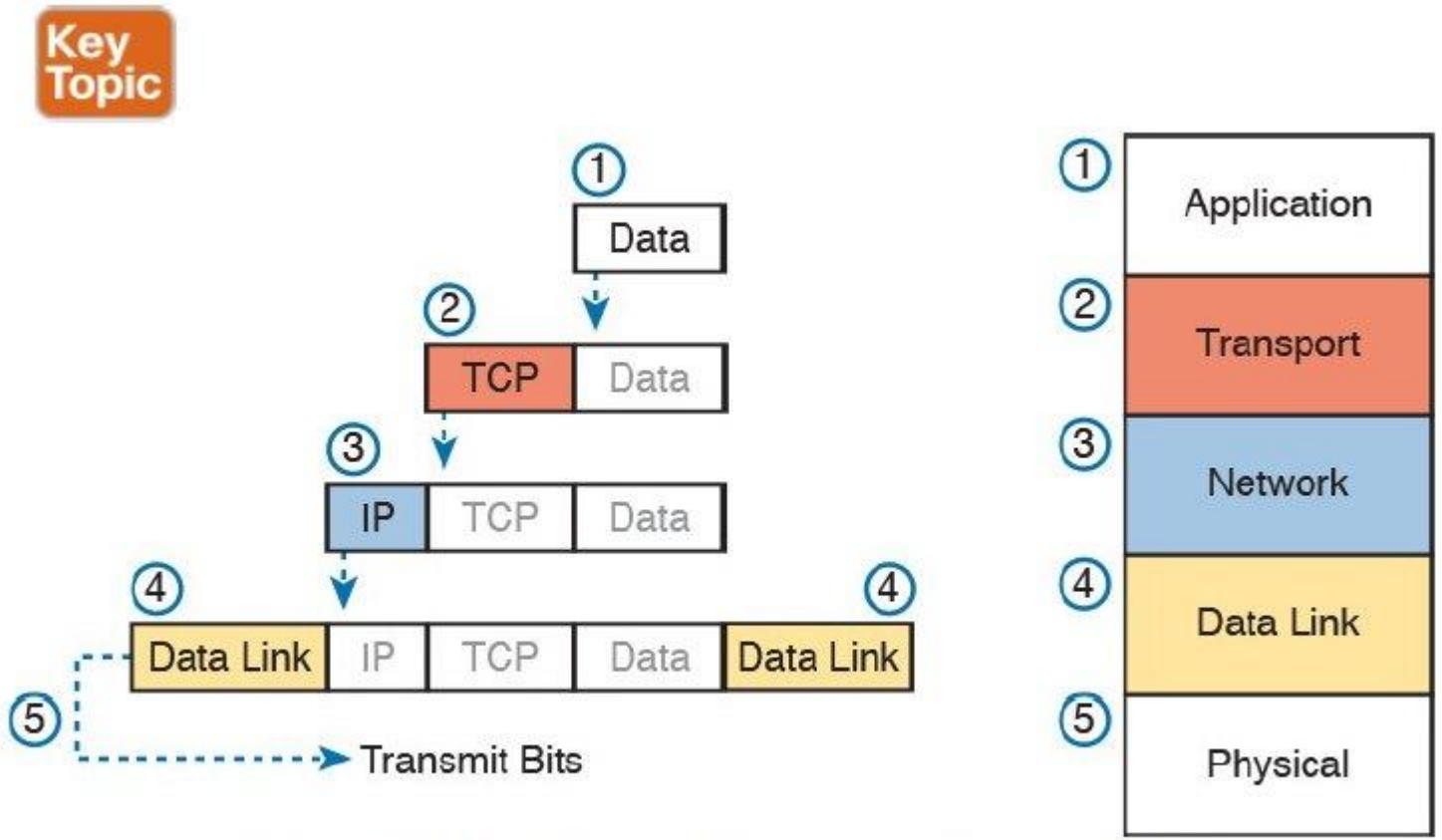
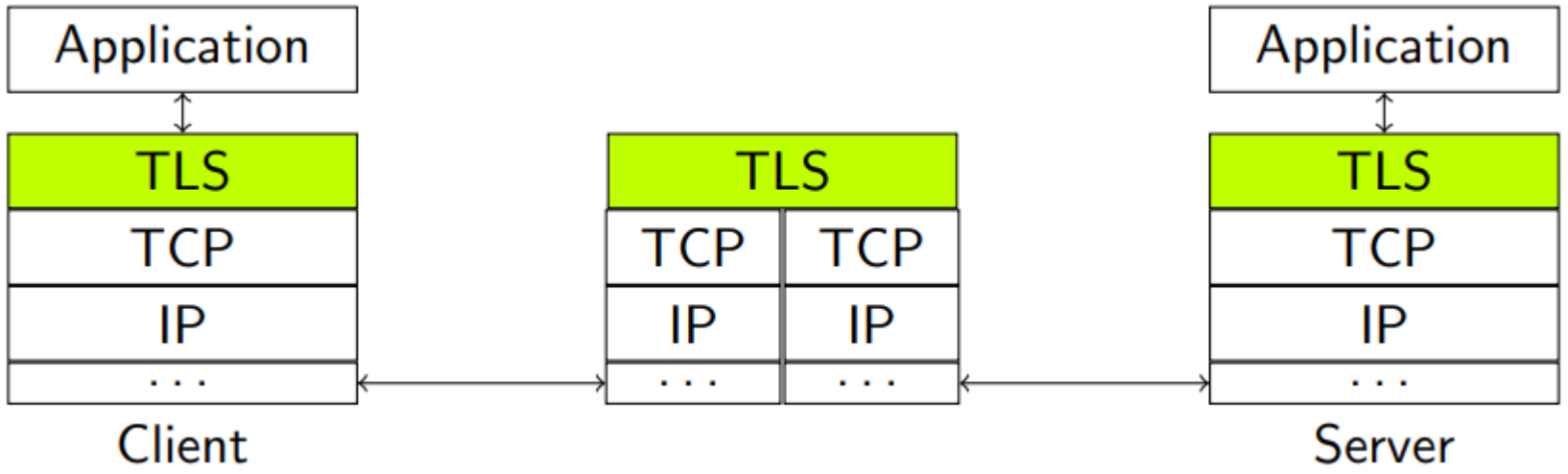
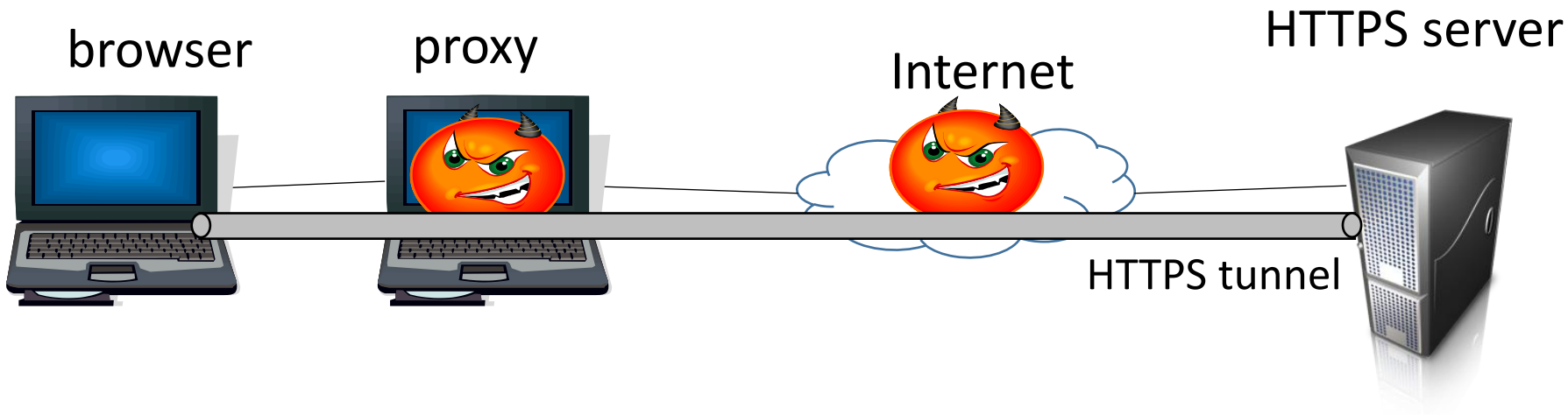


Figure 1-13 *Five Steps of Data Encapsulation: TCP/IP*

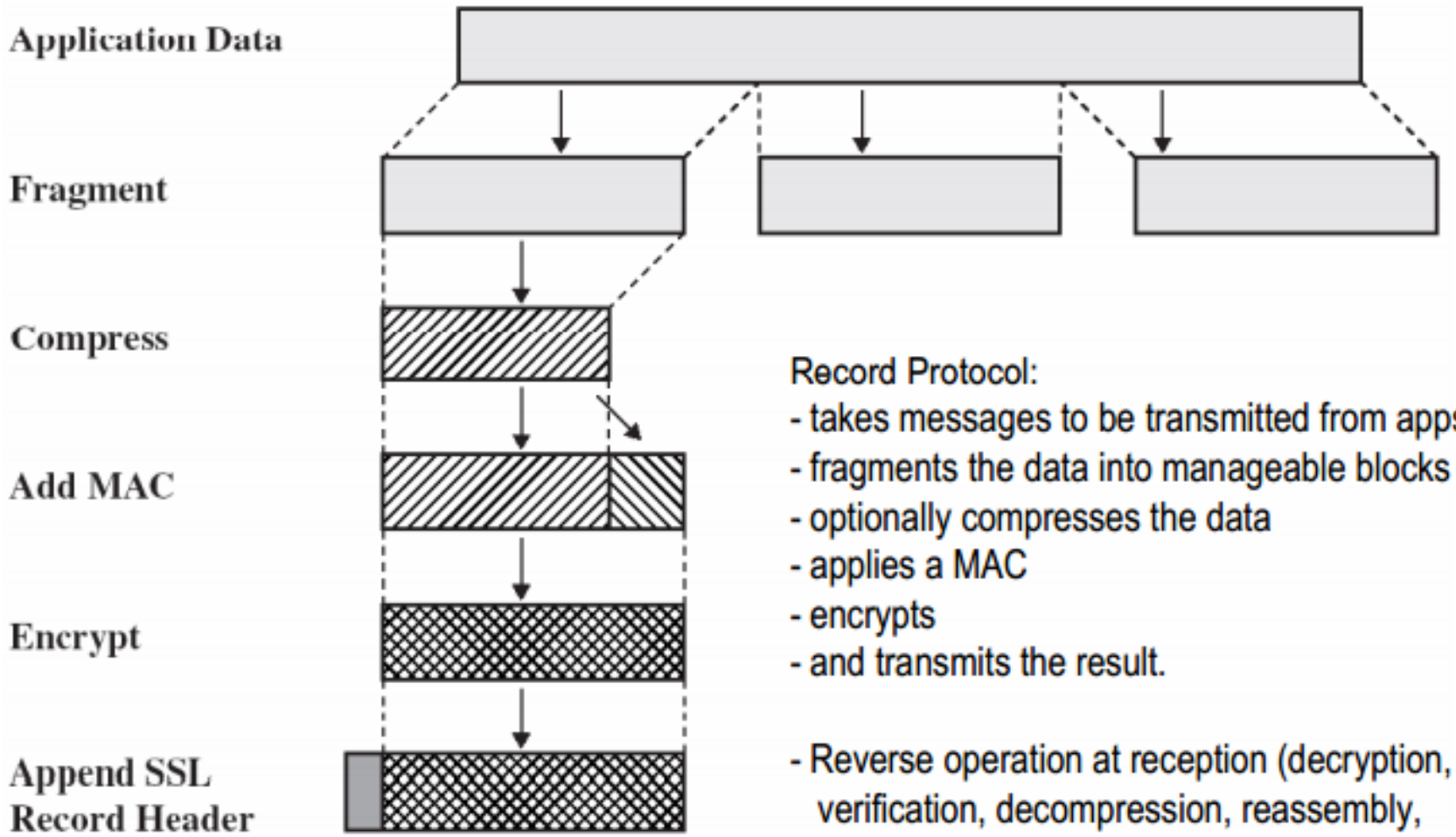
TCP/IP model and SSL/TLS



HTTPS: End to end secure protocol for http using TLS



Record Protocol operation



Record Protocol:
- takes messages to be transmitted from apps
- fragments the data into manageable blocks
- optionally compresses the data
- applies a MAC
- encrypts
- and transmits the result.

- Reverse operation at reception (decryption, verification, decompression, reassembly, delivery to apps)

TLS Handshake and the TLS record protocols

- The TLS protocol comprises two layers:

i) the TLS handshake

ii) and the TLS record protocol.

- The handshake protocol is executed first and it is responsible for authentication and key exchange and one more thing also (next slide!)

- The record protocol comes after handshake and its purpose is to encrypt using symmetric cryptography and integrity with HMAC.

- The record protocol uses **session keys** for security
  - Session keys are single-use keys

SSL/TLS

- TLS 1.2 supports many combinations of cipher suites (more than 30…)

TLS_RSA_WITH_3DES_CBC_MD5

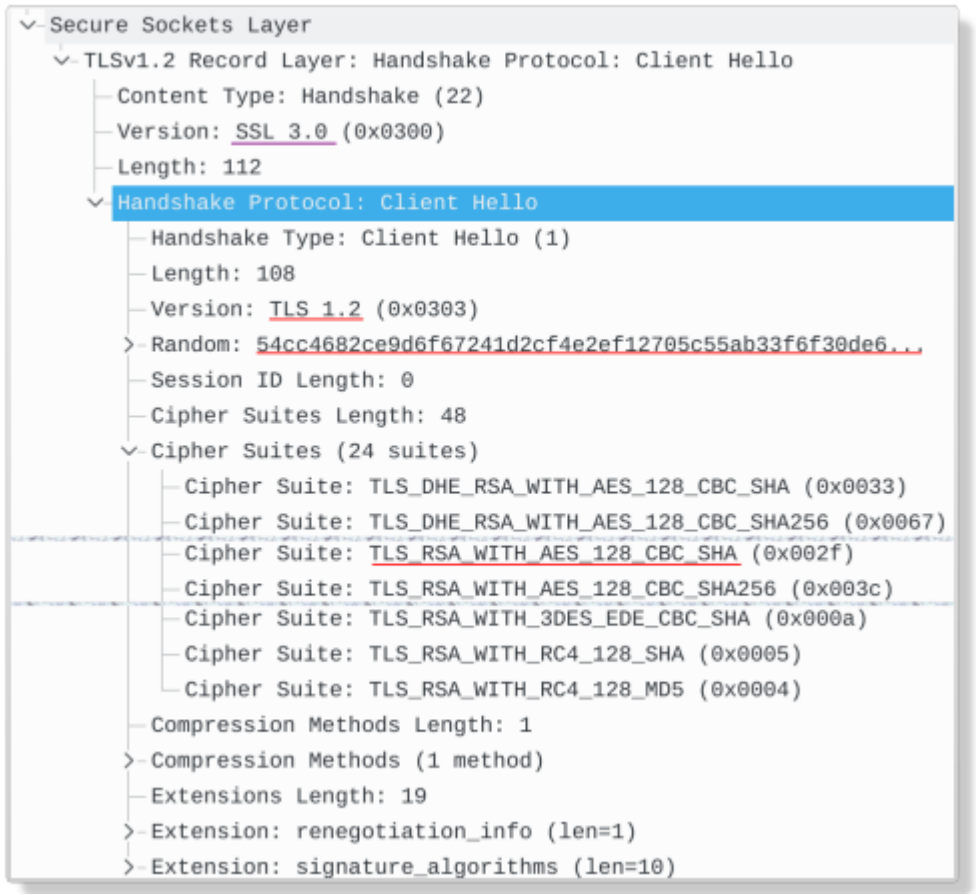
TLS_RSA_WITH_AES_256_CBC_SHA256

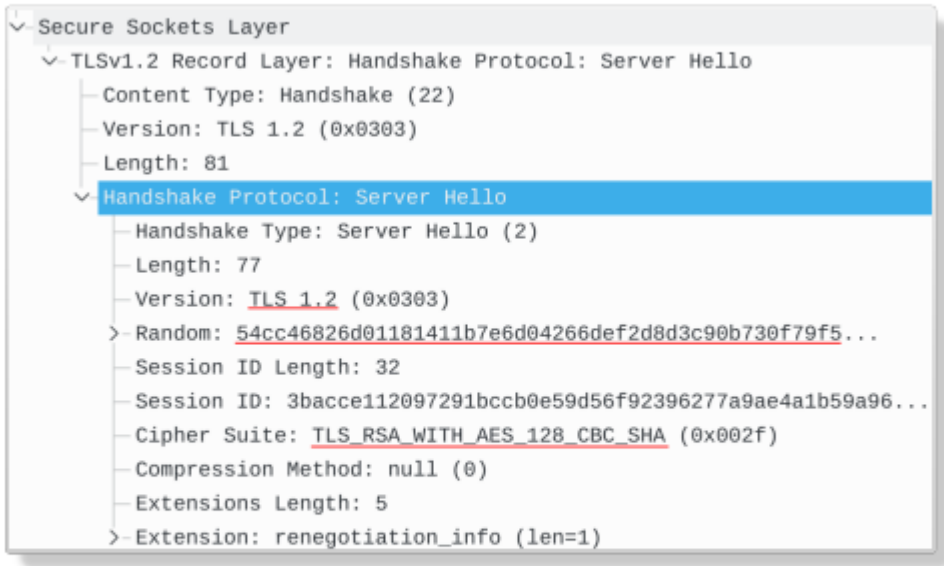
TLS_DHE_RSA_WITH_AES_256_CBC_SHA1

TLS_RSA_WITH_RC4_128_SHA

Q:How client and server decide the cipher suite to be used?

A: During the TLS handshake protocol!





+ Certificate (with RSA public key)
+ ServerHelloDone

TLS with RSA

TLS_RSA_WITH_AES_256_CBC_SHA256





TLS Handshake starts

Client Hello

Random Number A, supported ciphers

Client verifies certificate

Random Number B, selected ciphers, certificate

Server Hello

Client key exchange

K_M^+(Pre-master key)

Decrypt pre-master key with private key

Generate 4 keys using pre-master key

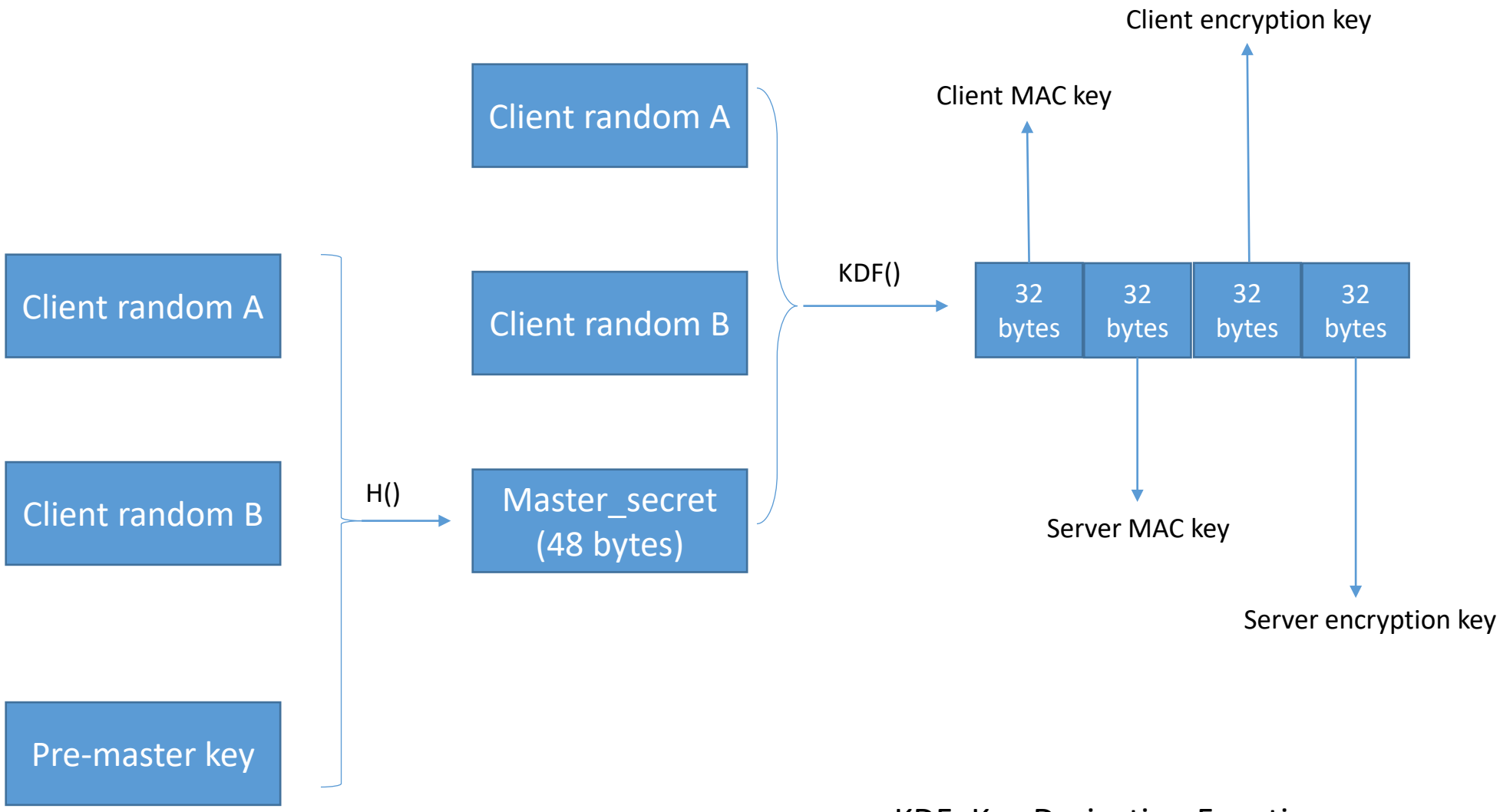
TLS Handshake ends

Generate 4 keys using pre-master key

TLS Record starts (encrypted and MAC protected

Data exchange (e.g,. HTML, img)

Key generation and expansion



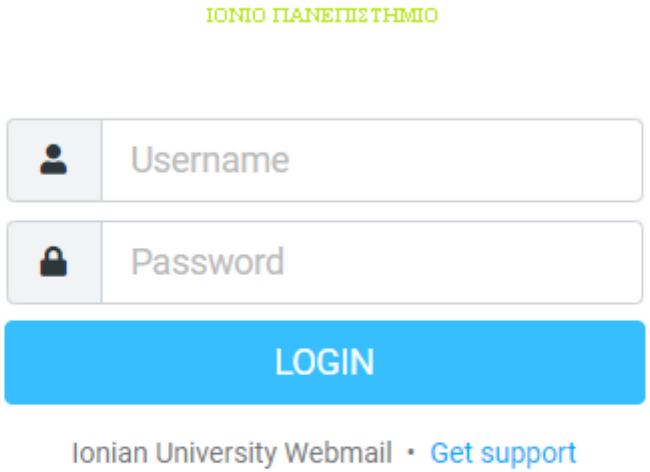
KDF=Key Derivation Function

TLS session keys vs Long term keys (definition)

- In general, a session key is used only once and its lifetime is the session itself
- In TLS, Client and Server have different symmetric session keys !!!
- Client and server have different symmetric session keys for HMAC and encryption!!
- A long term key is a key that remains the same between sessions (e.g., the private/public key of the server)

Client authentication

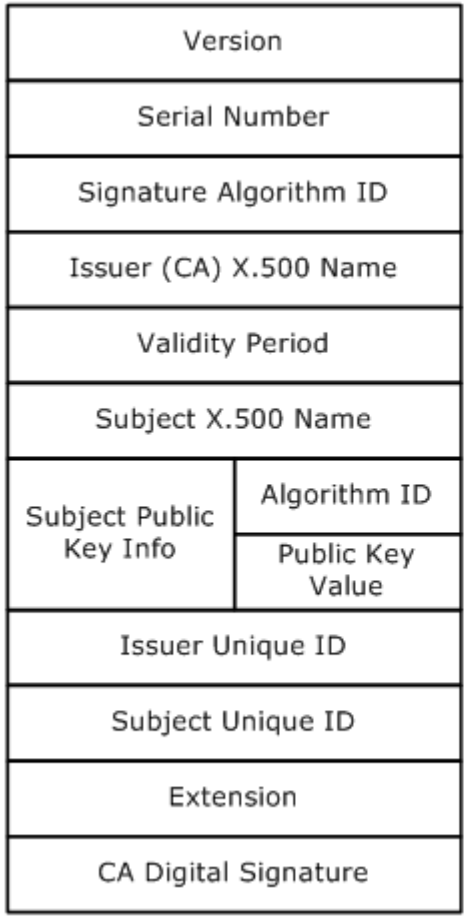
- What about client authentication?
- It is performed outside of TLS



TLS

- TLS client authentication is optional
- Clients usually do not have certificates!!
- Only valid cases are e-Government using an Idcard
- https://server.cryptomix.com/secure/

X.509 certificate format



Version of X.509 to which the Certificate conforms

A number that uniquely identifies the Certificate

The names of the specific Public Key algorithms that the CA has used to sign the Certificate (Ex.- RSA with SHA-1)

The identity of the CA Server who issued the Certificate

The period of time for which the Certificate is valid with start date and expiration date

The owner's identity with X.500 Directory format (Ex.- cn=auser, ou=SP, o=Alphawest)

The Public Key of the owner of the Certificate and the specific Public Key algorithms associated with the Public Key

Information used to identify the issuer of the Certificate

Information used to identify the Owner of the Certificate

Additional information like Alternate name, CRL Distribution Point (CDP)

The actual digital signature of the CA

Certificate verification

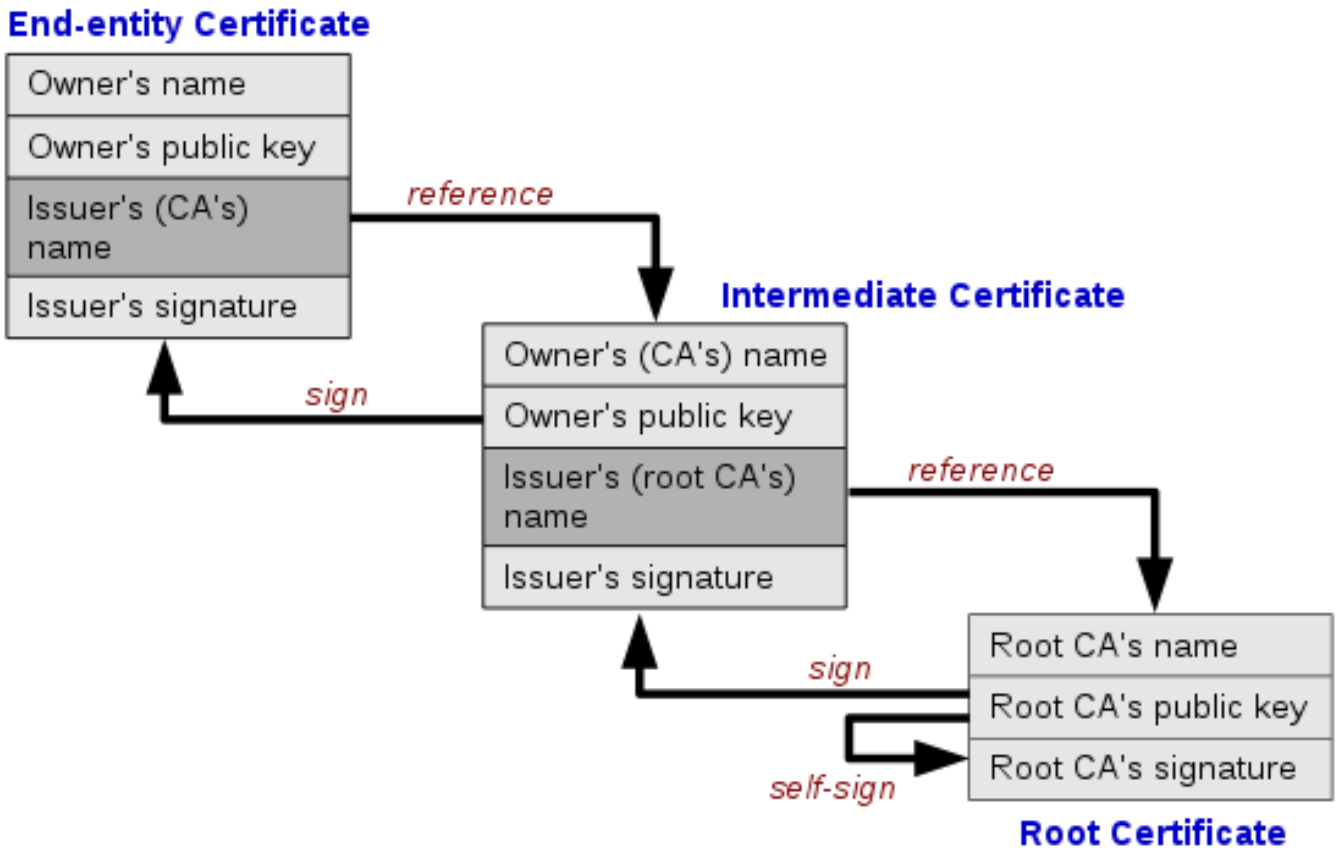
1. Signature validation (the whole chain – see next slide)

2. Certificate expiration

3. The most important check is whether CN == URL

- Who is going to perform this check?
  - The TLS library? The application (i.e., the browser)? The user?

Public Key Infrastructure (PKI)



What is the problem with TLS and RSA

- Imagine that the RSA private key of the web server is leaked.
- If someone recorded all message exchanges (TLS handshakes and TLS record data), then can he decrypt everything????
- The answer is yes!

Perfect Forward Secrecy

- A feature of specific key agreement protocols that gives assurances that session keys will not be compromised even if long-term secrets used in the session key exchange are compromised.

- The TLS RSA scheme does not have Perfect Forward Secrecy → It has been removed from TLS 1.2

- The Diffie-Hellman Key exchange has Perfect Forward Secrecy

TLS with Diffie-Hellman

TLS_DHE_RSA_WITH_AES_256_CBC_SHA256





TLS Handshake starts

Client Hello

Random Number A, supported ciphers

Random Number B, selected ciphers, certificate

Client verifies certificate and the signature

Server Hello

n, g, g^ymodn, K_M^-(g^ymodn, n, g)

Client key exchange

g^xmodn

pre-master key= g^{xy}modn

Pre-master key= g^{xy}modn

TLS Handshake ends

Generate 4 keys using pre-master key

Generate 4 keys using pre-master key

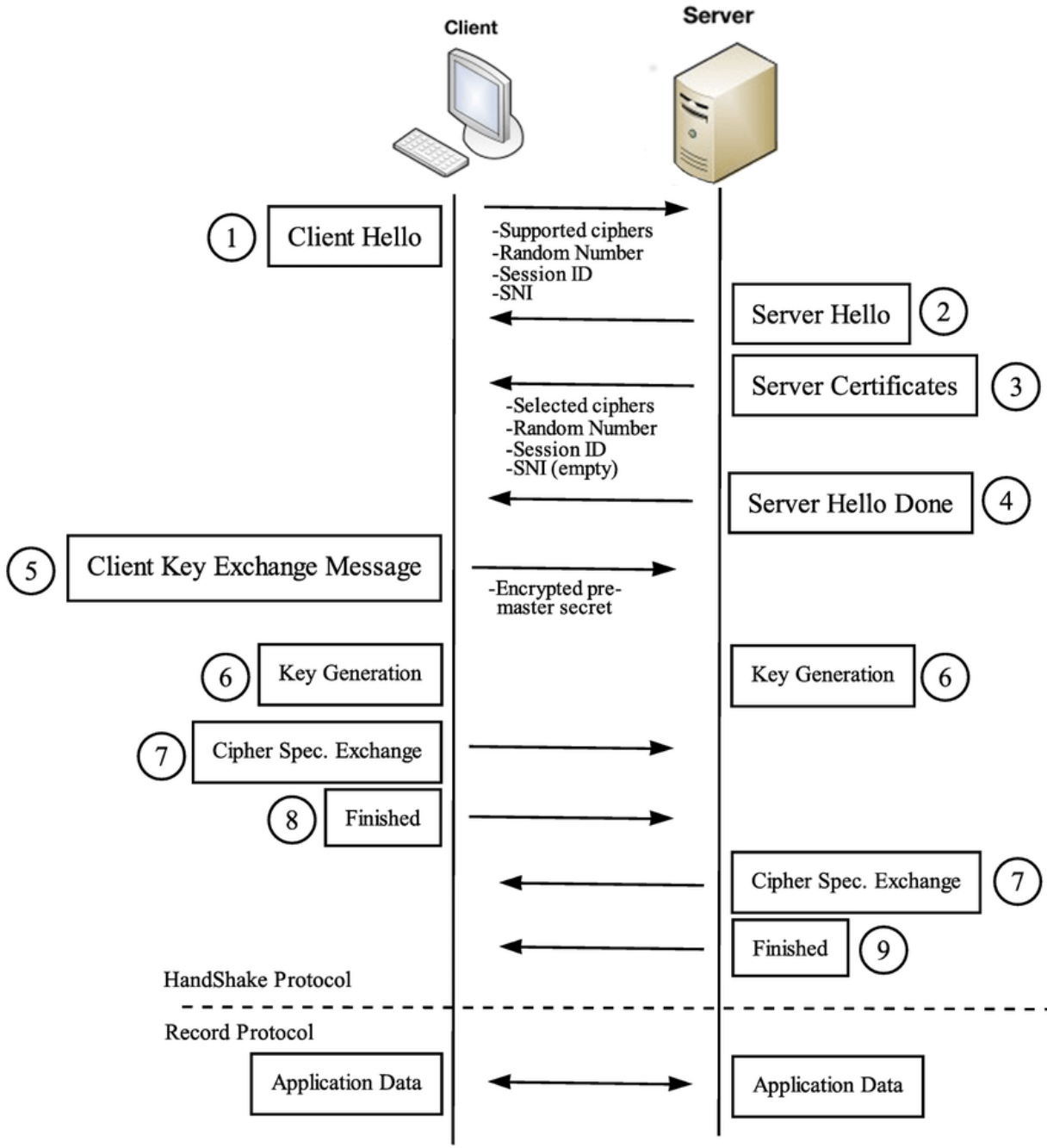
TLS Record starts

Data exchange (e.g,. HTML, img)

Open source implementations

- OpenSSL is the simplest form of CA and tool for PKI. It is a toolkit, developed in C, that is included in all major Linux distributions, and can be used both to build your own (simple) CA and to PKI-enable applications

- EJBCA is a full featured, Enterprise grade, CA implementation developed in Java. It can be used to set up a CA both for internal use and as a service.


Client
Server
1
Client Hello
-Supported ciphers
-Random Number
-Session ID
-SNI
Server Hello
2
Server Certificates
3
-Selected ciphers
-Random Number
-Session ID
-SNI (empty)
Server Hello Done
4
5
Client Key Exchange Message
-Encrypted pre-master secret
6
Key Generation
Key Generation
6
7
Cipher Spec. Exchange
8
Finished
Cipher Spec. Exchange
7
Finished
9
HandShake Protocol
Record Protocol
Application Data
Application Data

TLS 1.3

TLS: 1.3

- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
  - only 5 choices, rather than 37 choices
  - *requires* Diffie-Hellman (DH) for key exchange, rather than DH or RSA
  - combined encryption and authentication algorithm ("authenticated encryption") for data rather than serial encryption, authentication

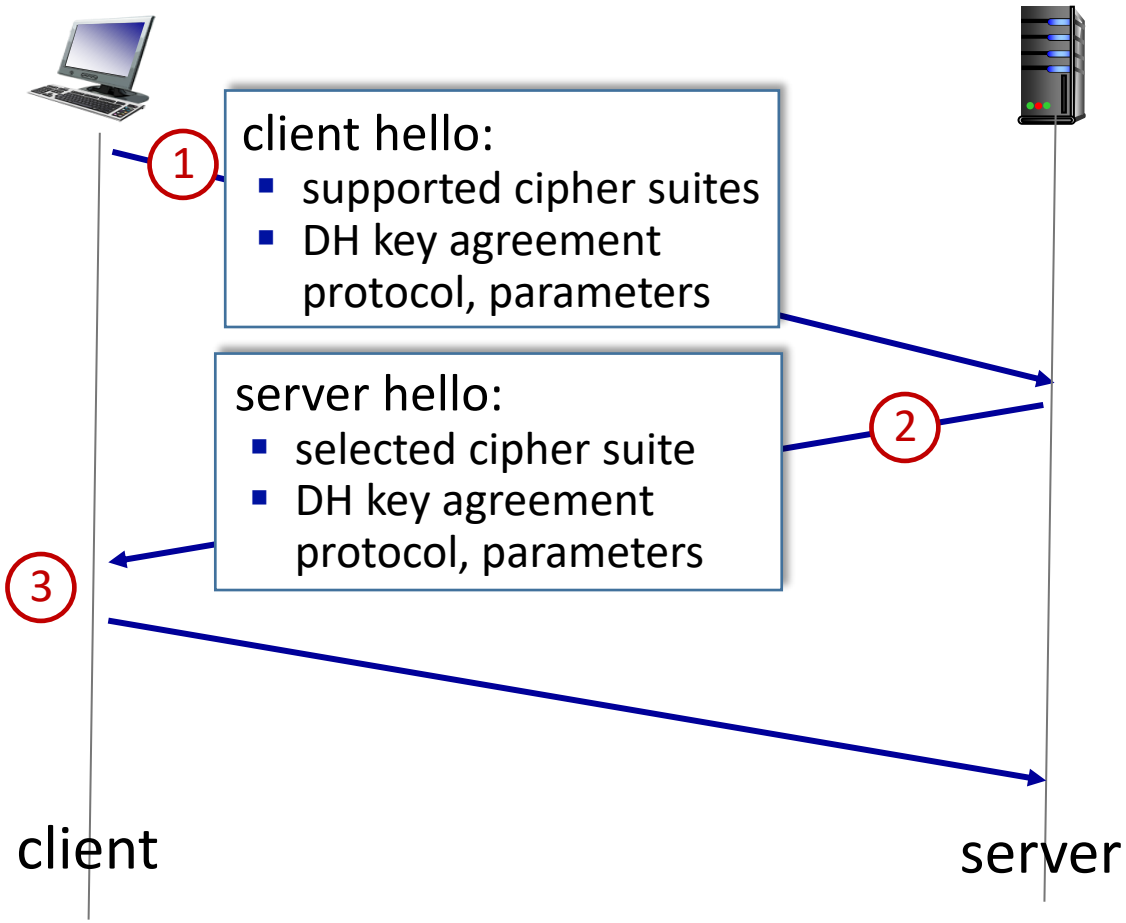
- Backwards compatible with TLS 1.2

TLS 1.3 cipher suites are 5 in total

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256

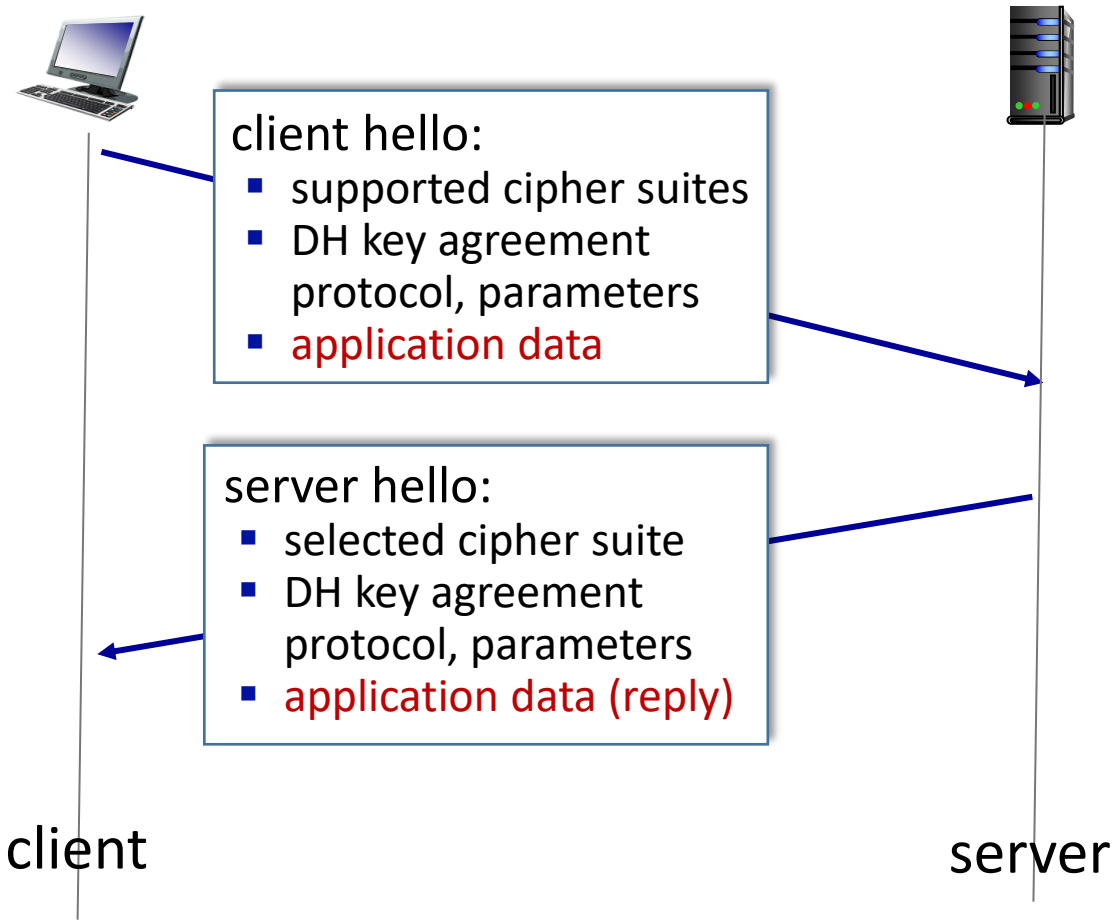
- Note that the authentication is not included as it is always (EC)DHE.
- Note that the last hash function is **not for HMAC but for KDF**
- All ciphers suites are AEAD (Authenticated encryption with associated data ()

TLS 1.3 handshake: 1 RTT



1. client TLS hello msg:
   - *guesses* key agreement protocol, parameters
   - indicates cipher suites it supports
2. server TLS hello msg chooses
   - key agreement protocol, parameters
   - cipher suite
   - server-signed certificate
3. client:
   - checks server certificate
   - generates key
   - can now make application request (e.g.., HTTPS GET)

TLS 1.3 handshake: 0 RTT



- initial hello message contains encrypted application data!
  - "resuming" earlier connection between client and server
  - application data encrypted using "resumption master secret" from earlier connection
- vulnerable to replay attacks!
  - maybe OK for get HTTP GET or client requests not modifying server state