CS 465
Computer Security

TLS

Fred Clift adapted from Daniel Zappala, Kent Seamons
Fall 2020

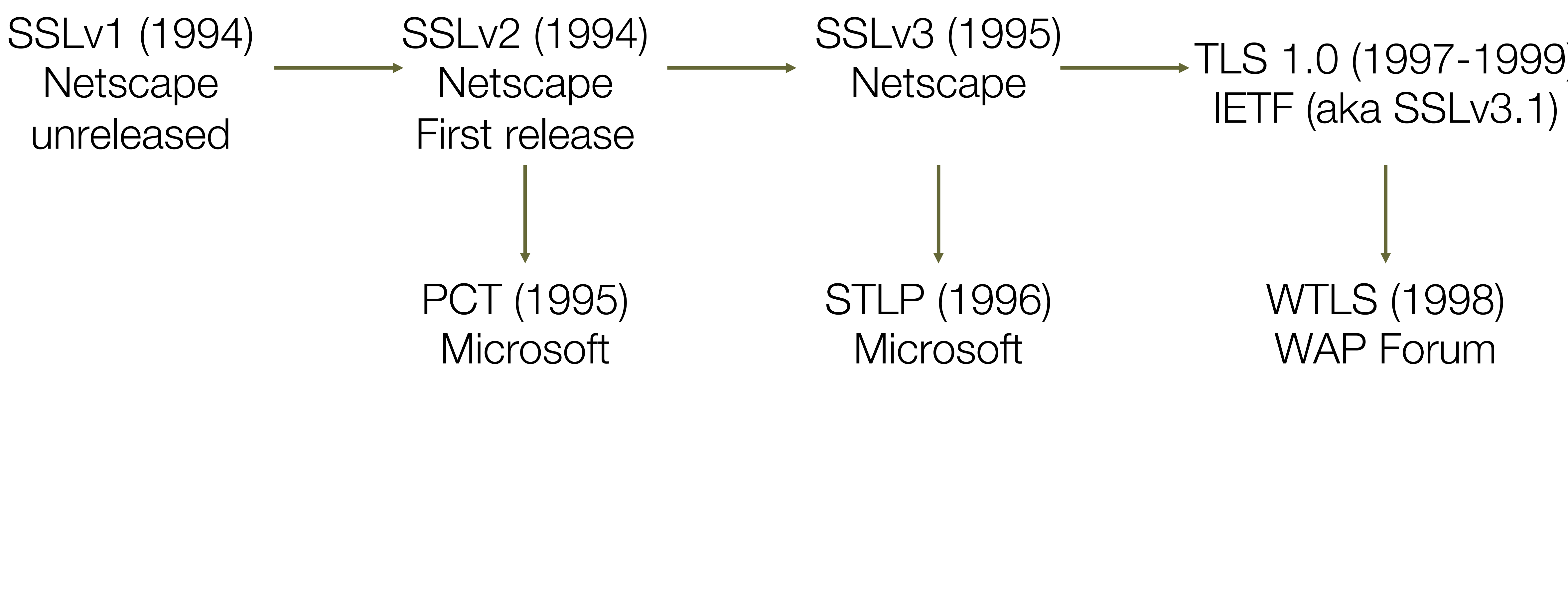
What Should You Learn From This Unit?

- Understand the TLS handshake
- Understand client/server authentication in TLS
  - RSA key exchange
  - DHE key exchange
  - Explain certificate ownership proofs in detail
  - What cryptographic primitives are used and why?
- Understand session resumption
- Understand the limitations of TLS
- Understand Forward Secrecy

History and Motivation

- Historic Network Protocols used no crypto
  - Telnet, Rsh , FTP, IRC, IMAP, POP, SMTP, SNMP and Later HTTP
  - “Secure Network” prior to TLS meant IPsec - point-to-point layer 3 traffic
    - not good for ‘any-to-any’
    - IPsec still in heavy use today
  - Secure Network Programming (SNP) - prototypes and papers
  - Early HTTP drove the real-world implementation of SSL, later renamed TLS – to be a “drop-in” replacement for sockets programming
  - Arbitrary Client/Server secure communications

Genesis of TLS



Source: SSL and TLS, Rescorla

Secure Sockets and Thread Models

STRIDE

- S Spoofing identity (either end)
- T Tampering (man-in-the-middle change the data in transit)
- R Repudiation
- I Information Disclosure
- D Denial of Service
- E Elevation of Privileges

Privacy, Authentication, Integrity. Forward Secrecy?

How would you do it?

Your goals:

- Privacy
- Authentication
- Data Integrity

Your Tools:

- Symmetric Ciphers
- Hashes
- Message Authentication Codes
- Key-"exchange" algorithms
- Public-Key Encryption
- Certificates and PKI

Queue Breakout Session 4 minutes

SNP/SSL/TLS Intro

Their goals:

- Privacy
- Authentication
- Data Integrity
- Provide Familiar Programming Interface
- Extensibility

Their Tools:

- Symmetric Ciphers
- Hashes
- Message Authentication Codes
- Key-”exchange” algorithms
- Public-Key Encryption
- Certificates and PKI

How'd we do?

Their goals:

- Privacy
  - Symmetric Encryption
  - RSA or DHE for passing keys
- Authentication
  - RSA/PKI
- Data Integrity
  - MAC
  - RSA or DHE for passing keys
- Provide Familiar Programming Interface - ”Socket” programming interface
- Extensibility – options built in to protocol

Now for some details

Protocols (and this protocol) specify lots of things:

- Record Formats
- On-the-wire Signaling
- Connection Setup/Termination Procedures
- Control Flow
- Error Conditions and Responses
- Data Encoding
- Compression?
- More?

We started at the motivation and view from 20,000 feet.
Now, lets go from the bottom up.

SSL Record Protocol Operation

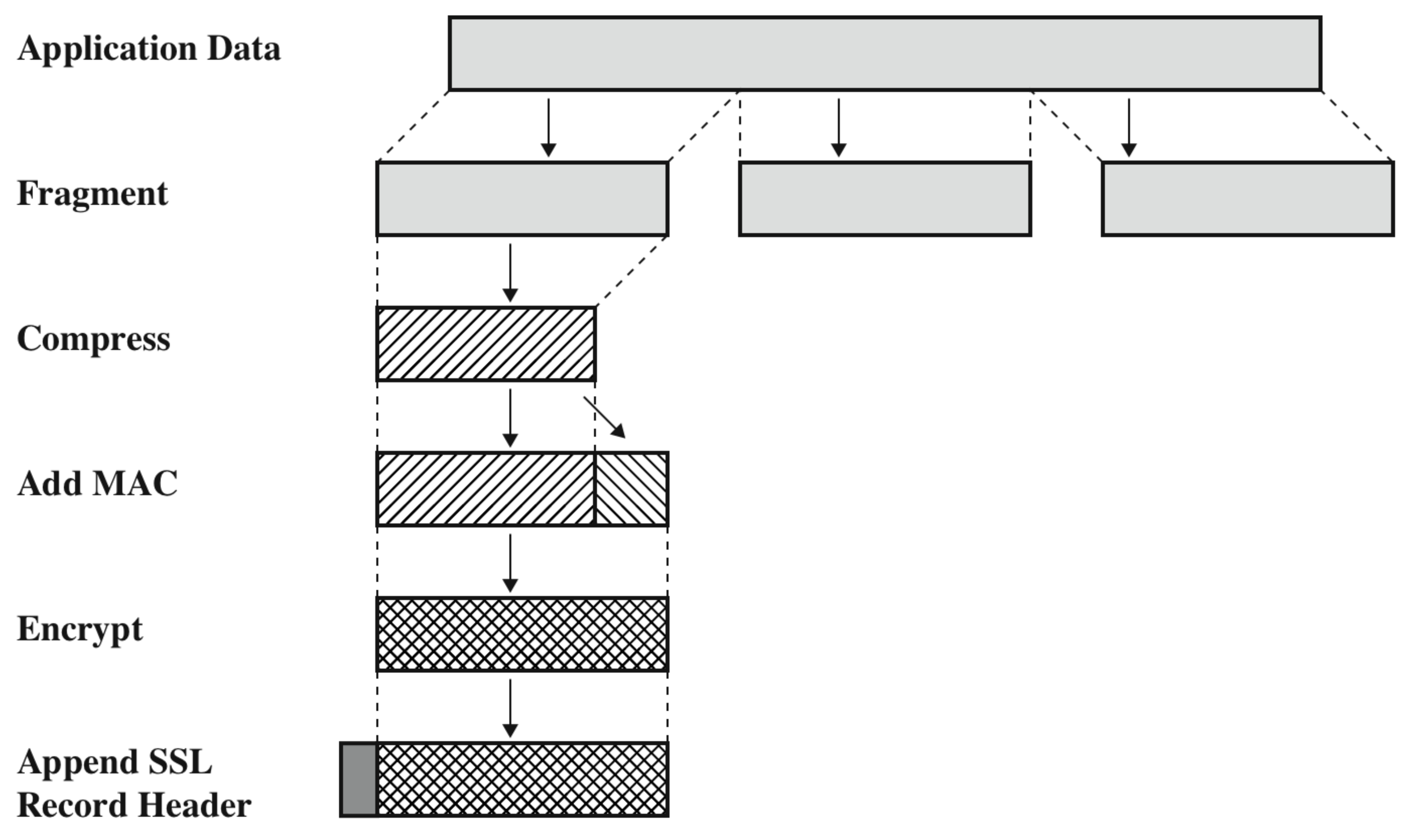


Figure 7.3 SSL Record Protocol Operation

Source: Network Security and Essentials (Stallings)

SSL Record Format

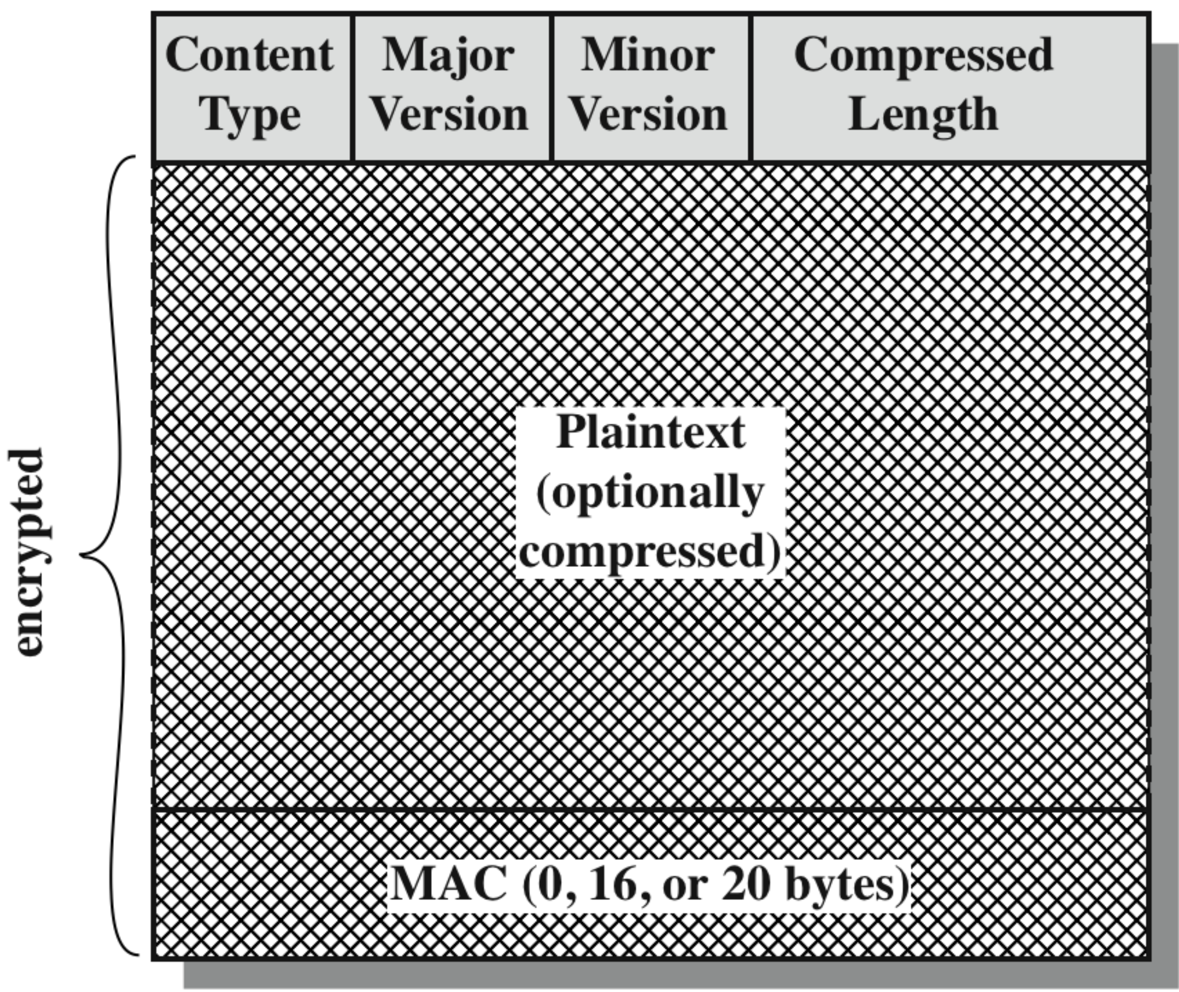
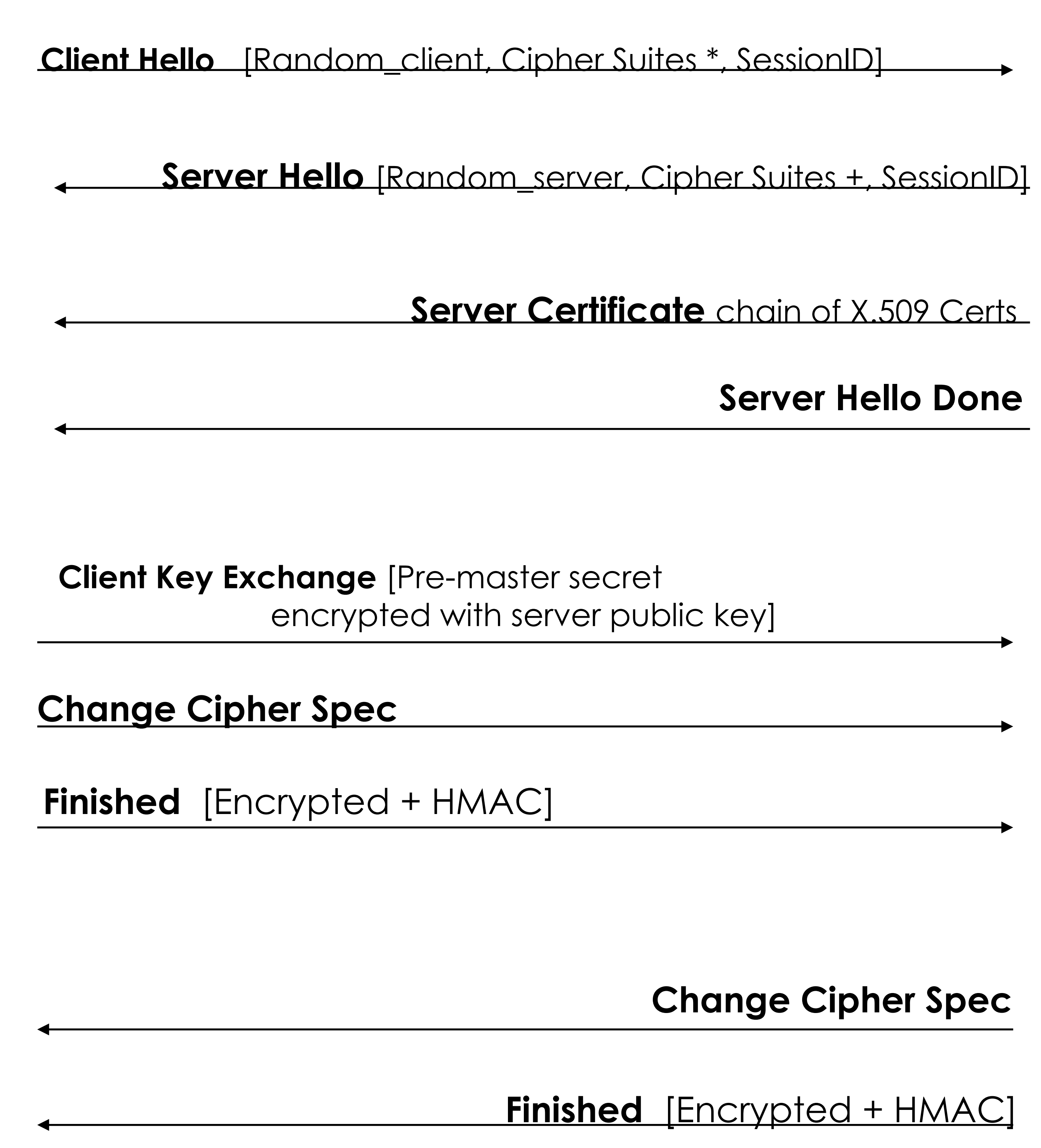


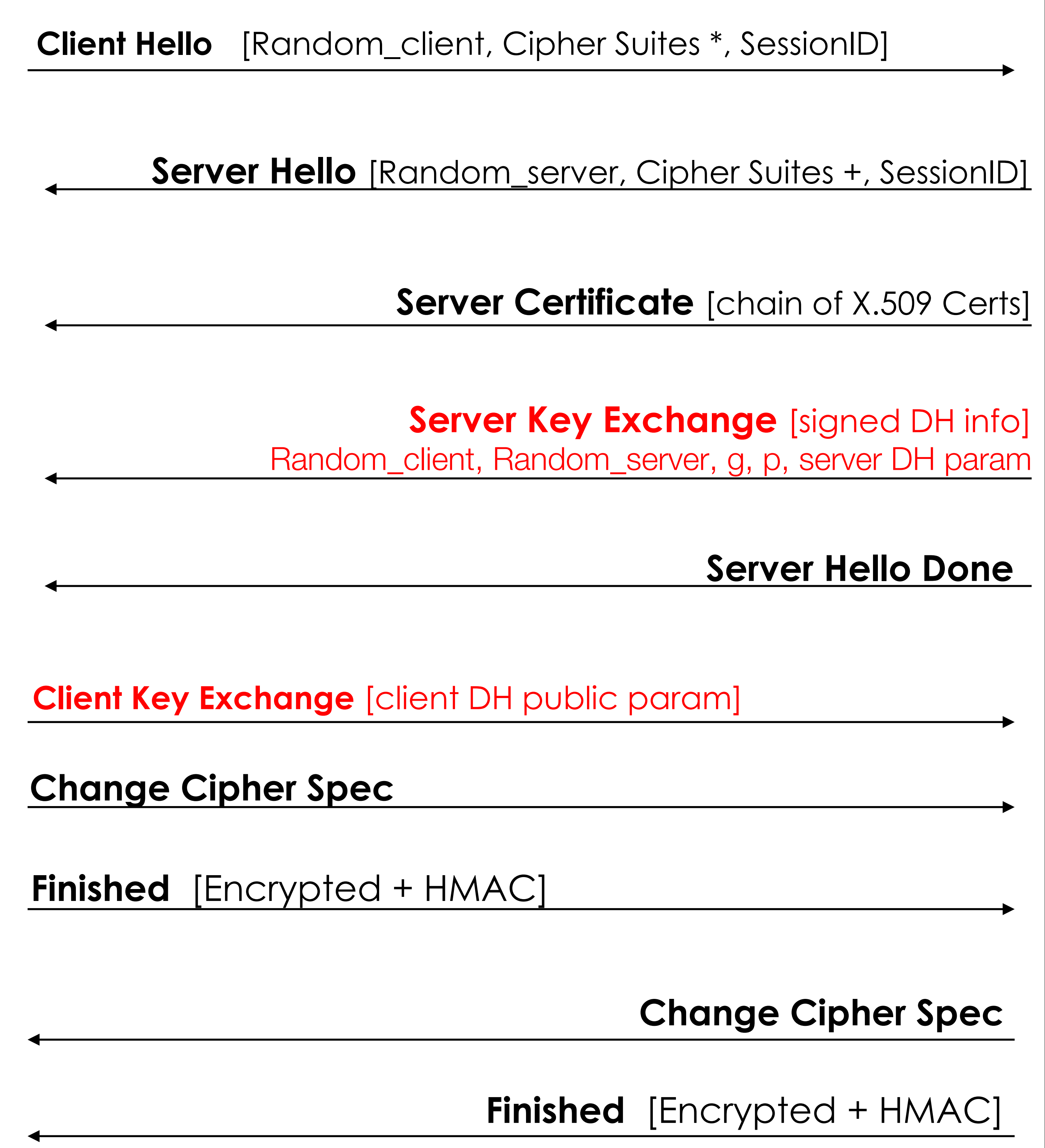
Figure 7.4 SSL Record Format

Source: Network Security and Essentials (Stallings)

RSA Key Exchange Method



DHE Key Exchange Method



Cipher Suite

- A Set of Algorithms (over 300 combinations supported)
- typically includes
  - key exchange algorithm (e.g. RSA, Diffie-Hellman)
  - bulk encryption algorithm (confidentiality, includes block cipher mode)
  - MAC algorithm (integrity)

  Like you had to have a specific name for every combination of appetizer, main-course, dessert at a restaurant

- examples
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

Cipher Suite Example

- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS protocol
  - DHE — Diffie-Hellman key exchange
  - RSA — authentication key — most commonly used
  - AES_128_GCM — symmetric, bulk encryption with 128 bit key, GCM mode
  - SHA256 — MAC algorithm

Easy right? How about DES-CBC-SHA?

Cipher Suite

- Must choose a safe cipher suite
  - Anonymous Diffie-Hellman (ADH) suites do not provide authentication
  - NULL cipher suites provide no encryption
  - Export cipher suites (limited to small key sizes that NSA can break) are insecure when negotiated in a connection, but they can also be used against a server that prefers stronger suites (the FREAK attack)
  - Suites with weak ciphers (typically of 40 and 56 bits) use encryption that can easily be broken
  - RC4 is insecure
  - 3DES is slow and weak

https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices

Cipher Suite

- An example configuration (order indicates preference)
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- Elliptic curve variations
  - ECDHE — Elliptic Curve Diffie-Hellman Ephemeral, key exchange algorithm — smaller keys for same security (elliptic curve cryptography) + ephemeral keys (forward secrecy)
  - ECDSA — Elliptic Curve Digital Signature algorithm, authentication algorithm — faster than RSA

https://scotthelme.co.uk/https-cheat-sheet/

Deriving the Master Secret and Keys

- generate a pre-master secret
  - a random number
  - it is REALLY HARD to generate a random number properly
- exchange the master secret, e.g. using RSA and padding
- derive the master secret using pre-master secret, the string "master secret", and the client and server random values
- generate keys using master secret (IV for each direction, symmetric key for each direction, MAC key for each direction)

Certificate Chain

- X.509 Certificates
  - standard format for digital certificates
- Chain
  - set of certificates that are signed, from server cert to intermediate certs to root cert
  - all the information needed to verify the server certificate
  - see prior lecture on Certificates

Finished

- The Finished message is the first one that is encrypted using the master secret
- The Finished message also includes an HMAC of (TLS <= 1.2 *part* of) the previously exchanged messages to ensure nobody tampered with the handshake

Perfect Forward Secrecy

- In vanilla RSA, the premaster secret is encrypted with the server's public key
  - If the server's private key is compromised all past and future sessions are also compromised
  - Majority of TLS < 1.3 uses vanilla RSA
- Using an ephemeral key
  - Even if the server's private key is later compromised, past sessions cannot be decrypted, even if captured and stored by a third party
  - Ephemeral Diffie-Hellman (DHE-RSA), Elliptic curve variation is faster (ECDHE)
  - No more deep packet inspection (worked by using RSA private key) for scanning/filtering software– replaced by proxies for some use cases

TLS is BROKEN (well not really)

- From https://en.wikipedia.org/wiki/Transport_Layer_Security_Security

Insert Rant about how every attack must have a cutsey name and website

4 Attacks against TLS/SSL
- 4.1 Renegotiation attack
- 4.2 Downgrade attacks: FREAK attack and Logjam attack
- 4.3 Cross-protocol attacks: DROWN
- 4.4 BEAST attack
  - 4.4.1 CRIME and BREACH attacks
- 4.5 Timing attacks on padding
- 4.6 POODLE attack
- 4.7 RC4 attacks
- 4.8 Truncation attack
- 4.9 Unholy PAC attack
- 4.10 Sweet32 attack
- 4.11 Implementation errors: Heartbleed bug, BERserk attack, Cloudflare bug
- 4.12 Survey of websites vulnerable to attacks

TLS 1.3 Updates

- Reduce Handshake Round Trips/Latency
- Remove Legacy Features
  - Deprecated ciphersuites (Only AEAD cipher modes allowed)
  - No RSA key exchange (Forward Secrecy)
  - Fixed set of DH parameters
  - No CBC modes (re: mac-then-encrypt/padding-oracle-attacks and friends)
  - No Compression
  - No Unixtime
- Only 5 ciphersuites (cipher, kx, signature are separate negotiations now)
  - To wrap our heads around it we can think of them like this
  - TLS_AES_256_GCM_SHA384
  - TLS_CHACHA20_POLY1305_SHA256
  - TLS_AES_128_GCM_SHA256
  - TLS_AES_128_CCM_8_SHA256
  - TLS_AES_128_CCM_SHA256

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

https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a/

TLS 1.3 Updates (continued)

- Ciphersuites continued:

  One mail goal of TLS 1.3 was to move to ONLY AEAD modes.

  Cryptographers and Programmers do the wrong thing with simpler primitives

  No “pick your block mode, pick your MAC, and then combine them” – instead specify a “mode” that takes all the choice and guesswork away. So, blockmode and MAC are intrinsic parts of the AEAD modes that TLS 1.3 uses.

- Encrypt /MAC the Handshake – downgrade attacks like FREAK
- Fixed RSA padding (RSA-PSS vs PKCS#1v1.5)
- Mitigate/Improve defence re: Cross Protocol Attacks
  - TLS <= 1.2 allowed a client to get the server to sign a message with a chosen 32-byte prefix with some possible theoretical attacks

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

https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a/

Session Resumption

- In a TLS connection, the SessionID can be null, indicating a new connection
- A non-null SessionID means the client would like to resume a prior session
  - avoid full handshake (e.g. avoid expensive public key crypto operations and extra round trips)
  - this gets complex if the server is distributed across multiple machines, see for example https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/
  - session resumption also messes with forward secrecy, see for example https://blog.compass-security.com/2017/06/about-tls-perfect-forward-secrecy-and-session-resumption/
- Can use session tickets instead (see RFC 5077)
  - these are also problematic, see for example https://blog.filippo.io/we-need-to-talk-about-session-tickets/

Client Authentication

- server must request it
- a person must purchase a certificate
  - most people have no idea what a cert is
  - usually involves manual verification of identity (if cert is tied to some personal identifier)
  - expensive and time-consuming relative to Let’s Encrypt
  - must protect private key
- a person must configure their browser to use the certificate and select it when prompted by the browser
  - those interfaces are *not* pretty

RSA Key Exchange Method

Client Authentication

Client | Server

Client Hello [Random_client, Cipher Suites *, SessionID] →

← **Server Hello** [Random_server, Cipher Suites +, SessionID]

← **Server Certificate** chain of X.509 Certs

← **Client Certificate Request**

← **Server Hello Done**

Certificate →

Client Key Exchange [Pre-master secret encrypted with server public key] →

Certificate Verify →

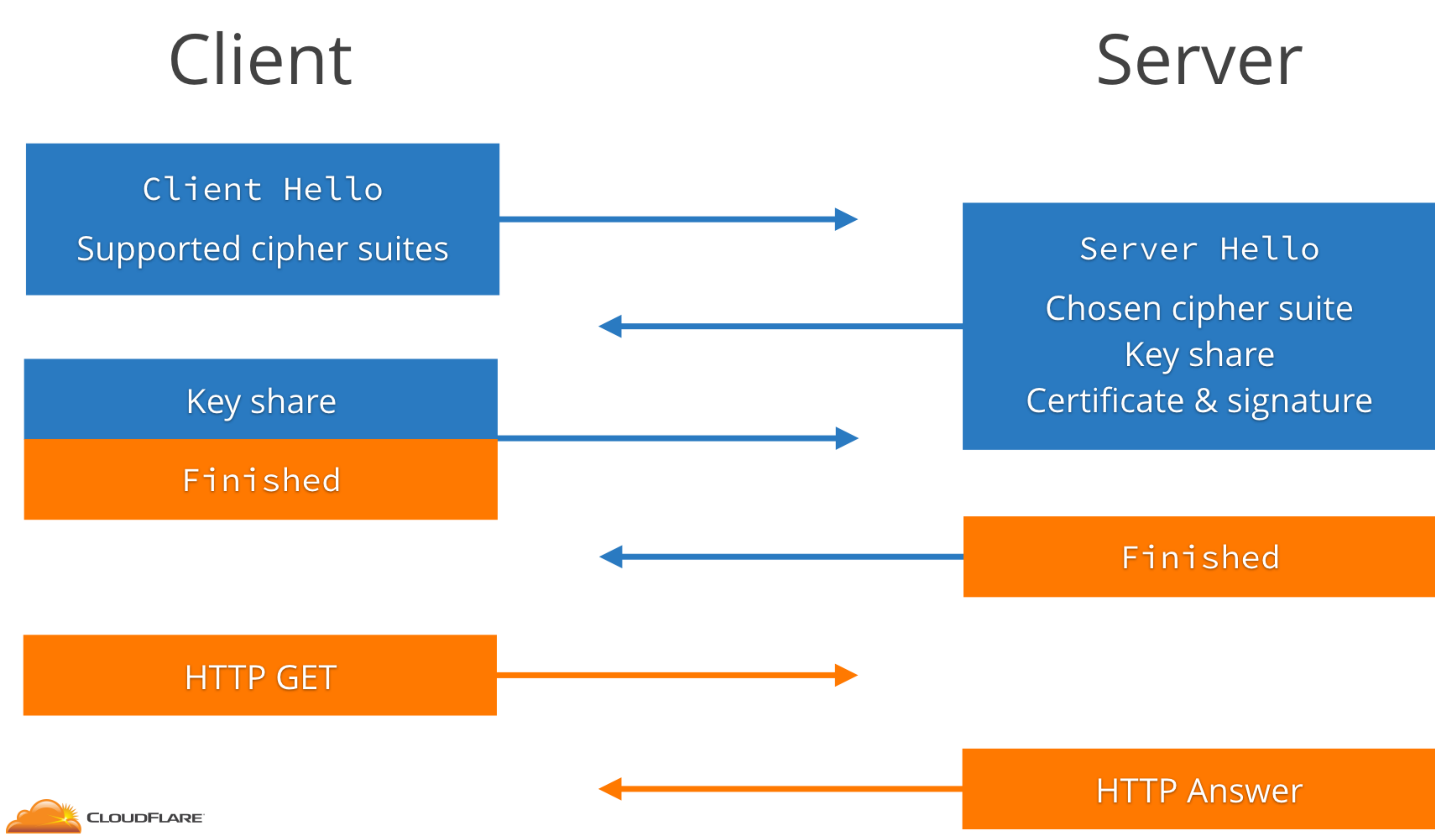
Change Cipher Spec →

Finished [Encrypted + HMAC] →

← **Change Cipher Spec**

← **Finished** [Encrypted + HMAC]

TLS 1.2 (Simplified)



TLS 1.3 (Simplified)

Client

Server

`Client Hello`

Supported cipher suites
Key share

`Server Hello`

Chosen cipher suite
Key share

Certificate & signature

`Finished`

`Finished`

HTTP GET

HTTP Answer

TLS 1.2 Resumption

Client

Server

`Client Hello`

Session ID / Ticket

`Server Hello`

`Finished`

`Finished`

HTTP GET

HTTP Answer

TLS 1.3 Resumption (0-RTT)

Client

Server

Client Hello

Session Ticket (PSK)
Key share

HTTP GET

Server Hello

Key share

Finished

HTTP Answer

TLS 1.3 Resumption (0-RTT)

- Beware
  - 0-RTT data is not forward secret — if an attacker gets a session ticket key at some point, they can decrypt this data
    - servers need to rotate session ticket keys frequently
    - still an improvement over TLS 1.2 session tickets
  - subject to replay attacks
    - *The solution is that servers must not execute operations that are not idempotent received in 0-RTT data.*
    - *E.g. limit 0-RTT data to a an HTTP GET*

Review Questions

- How many shared keys are derived between a client and a server that establish a TLS session?
- How does the server prove ownership of its private key?
- How does the client prove ownership of its private key when client authentication is (rarely) used?
- What is the pre-master secret?
  - Who creates it?
  - How is it securely transmitted?
- What is session resumption?
  - How does it differ from a regular SSL handshake?
- When do the client and server start encrypting traffic using symmetric encryption?

Review Questions

- How many shared keys are derived between a client and a server that establish a TLS session?
  - Each side generates 4-6 keys
- How does the server prove ownership of its private key?
  - Implicitly by decrypting the pre-master secret and finishing handshake
- How does the client prove ownership of its private key when client authentication is (rarely) used?
  - Send digital signature to the server
- What is the pre-master secret?
  - Who creates it?
  - How is it securely transmitted?
- What is session resumption?
  - How does it differ from a regular SSL handshake?
- When do the client and server start encrypting traffic using symmetric encryption?
  - Finished message