CS 465
Computer Security

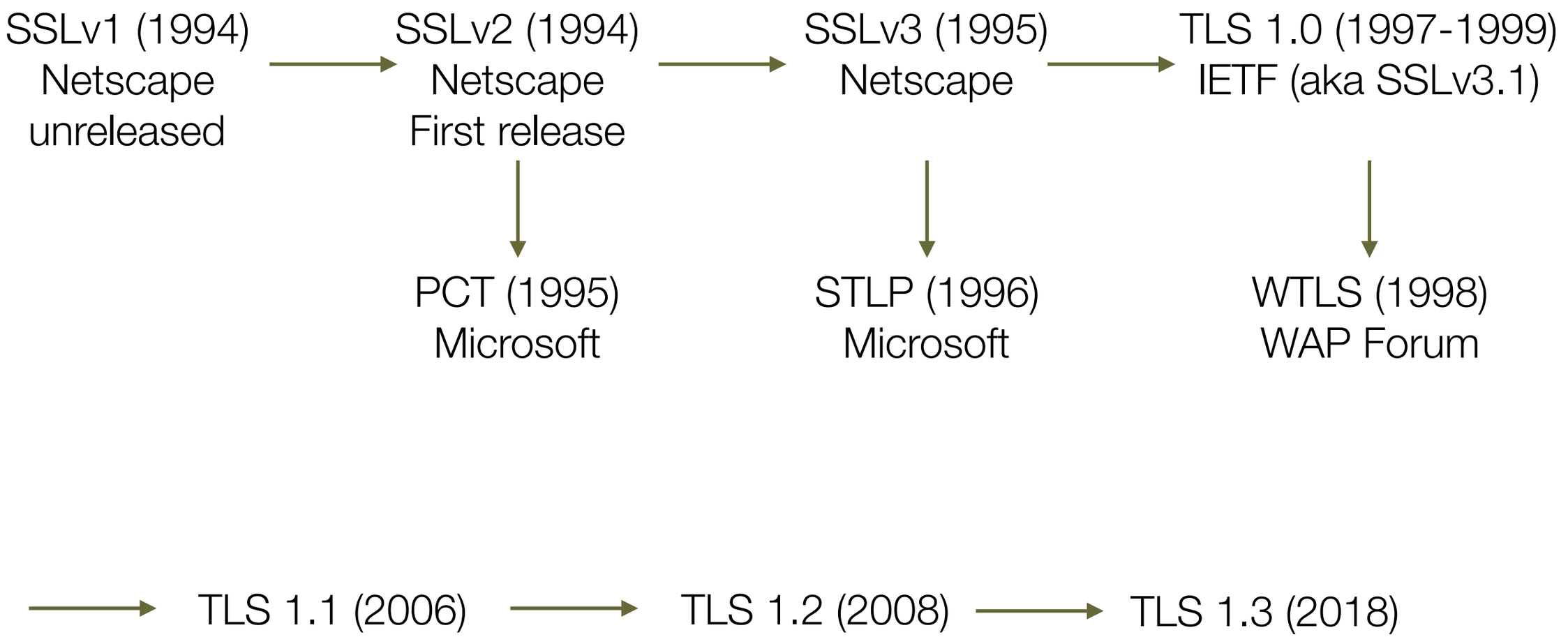
TLS

Daniel Zappala, adapted from Kent Seamons
Fall 2018

Goals

- 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

Genesis of TLS



Source: SSL and TLS, Rescorla

SSL Record Protocol Operation

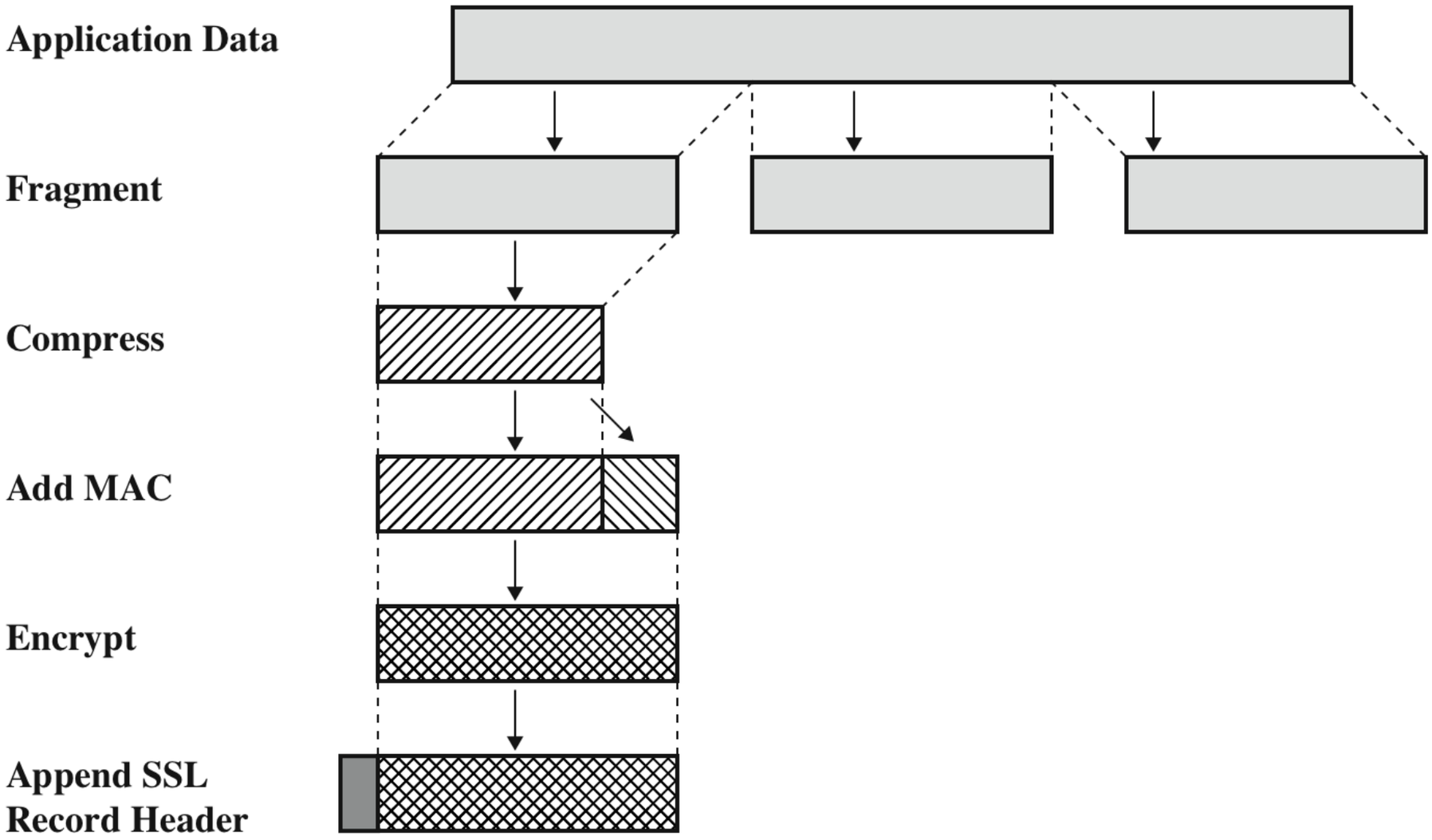


Figure 7.3 SSL Record Protocol Operation

Source: Network Security and Essentials (Stallings)

SSL Record Format

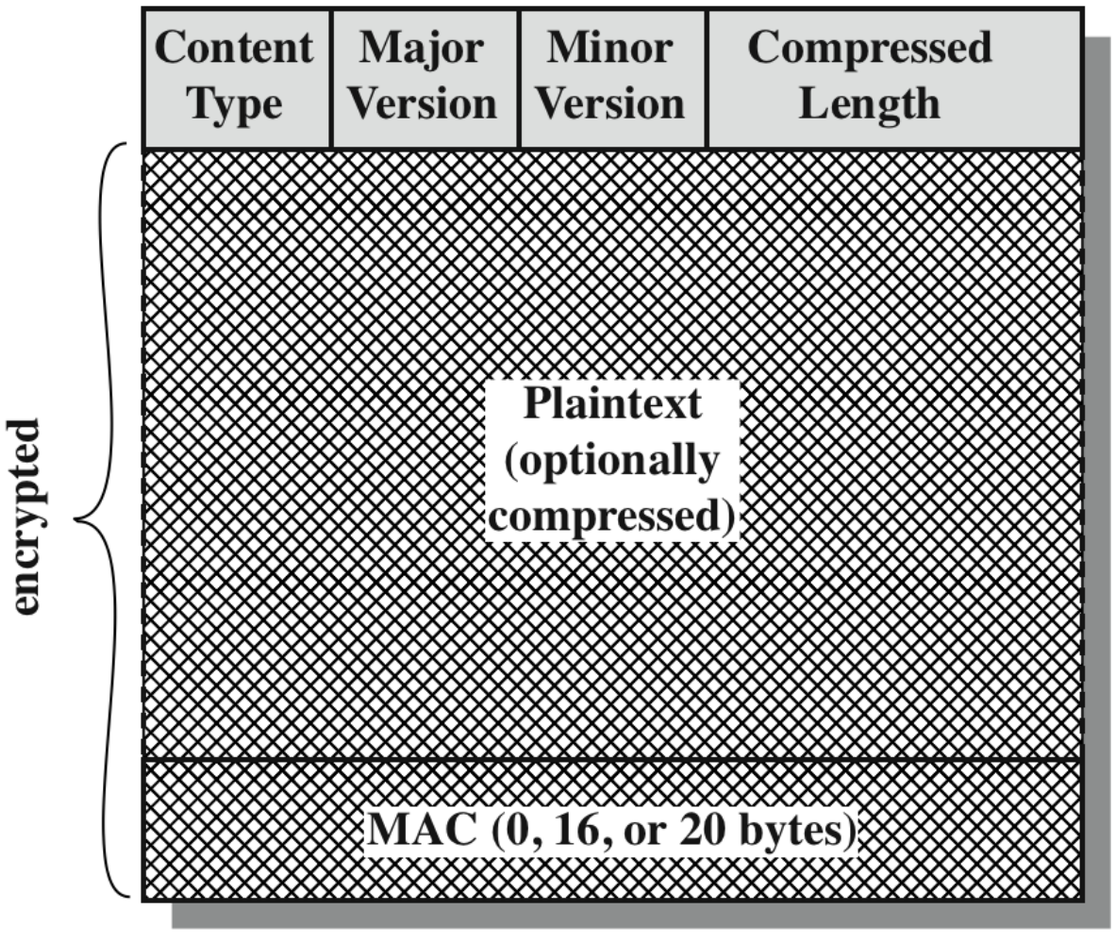
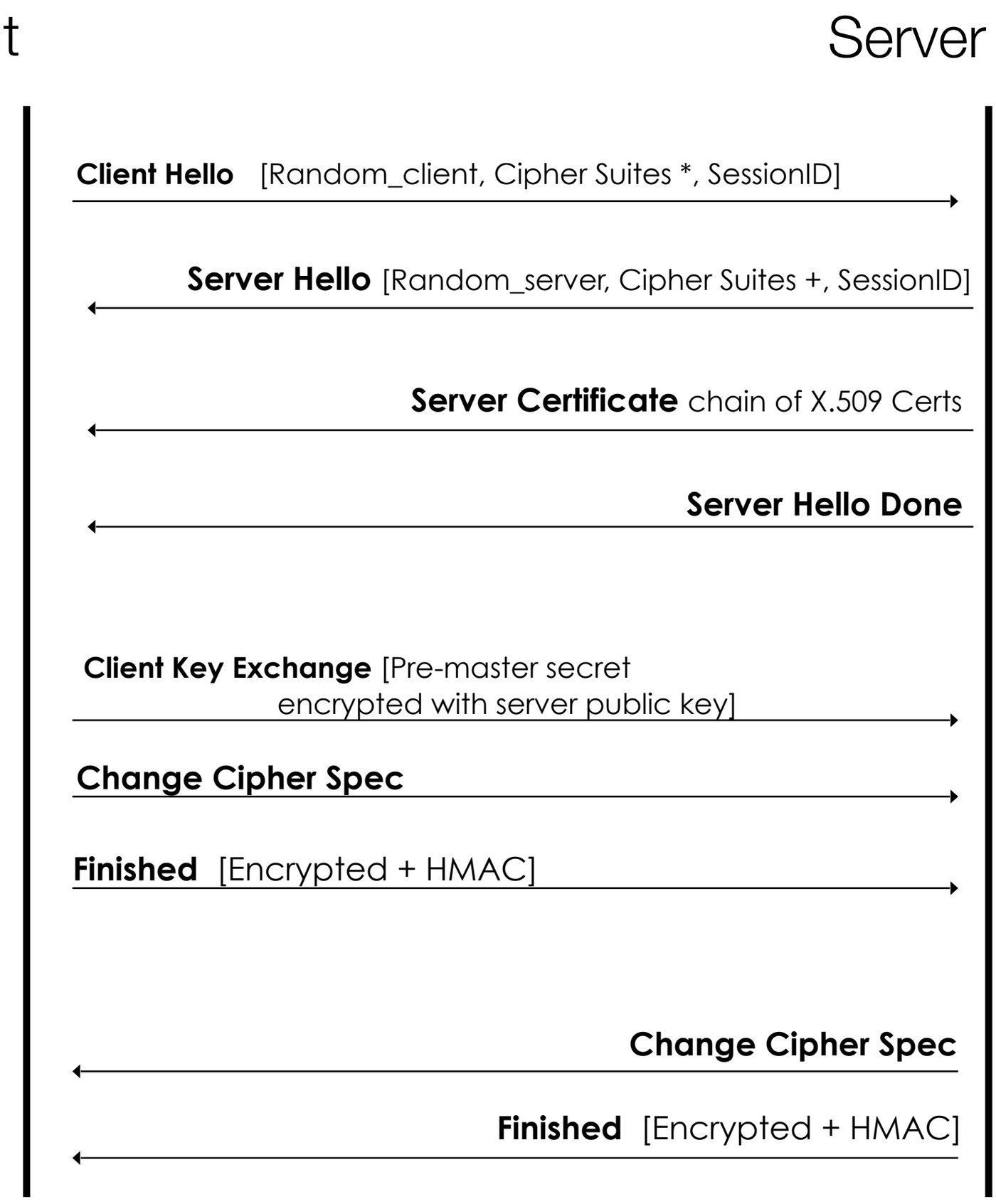


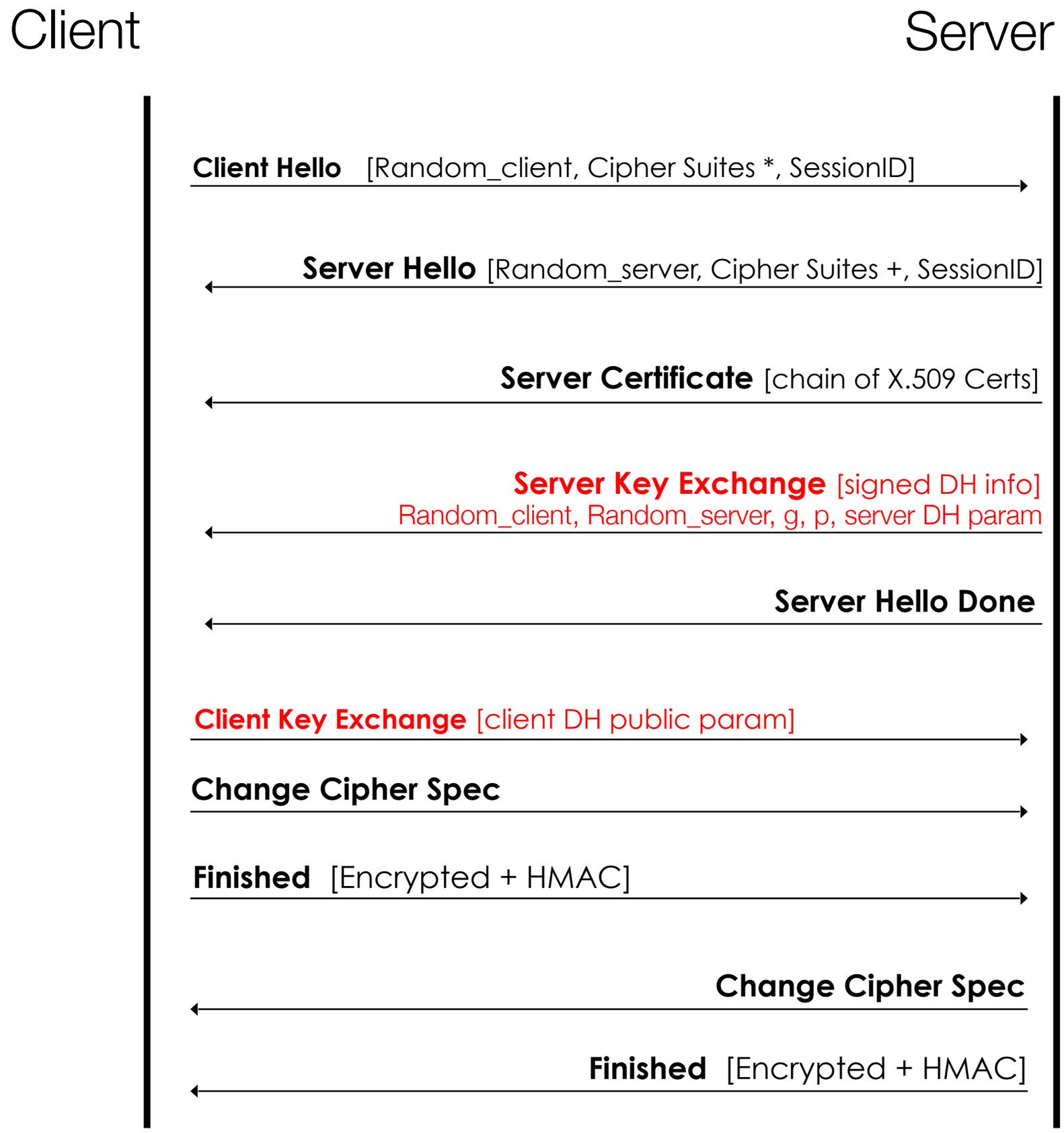
Figure 7.4 SSL Record Format

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)
- examples
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

Cipher Suite

- 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

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

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

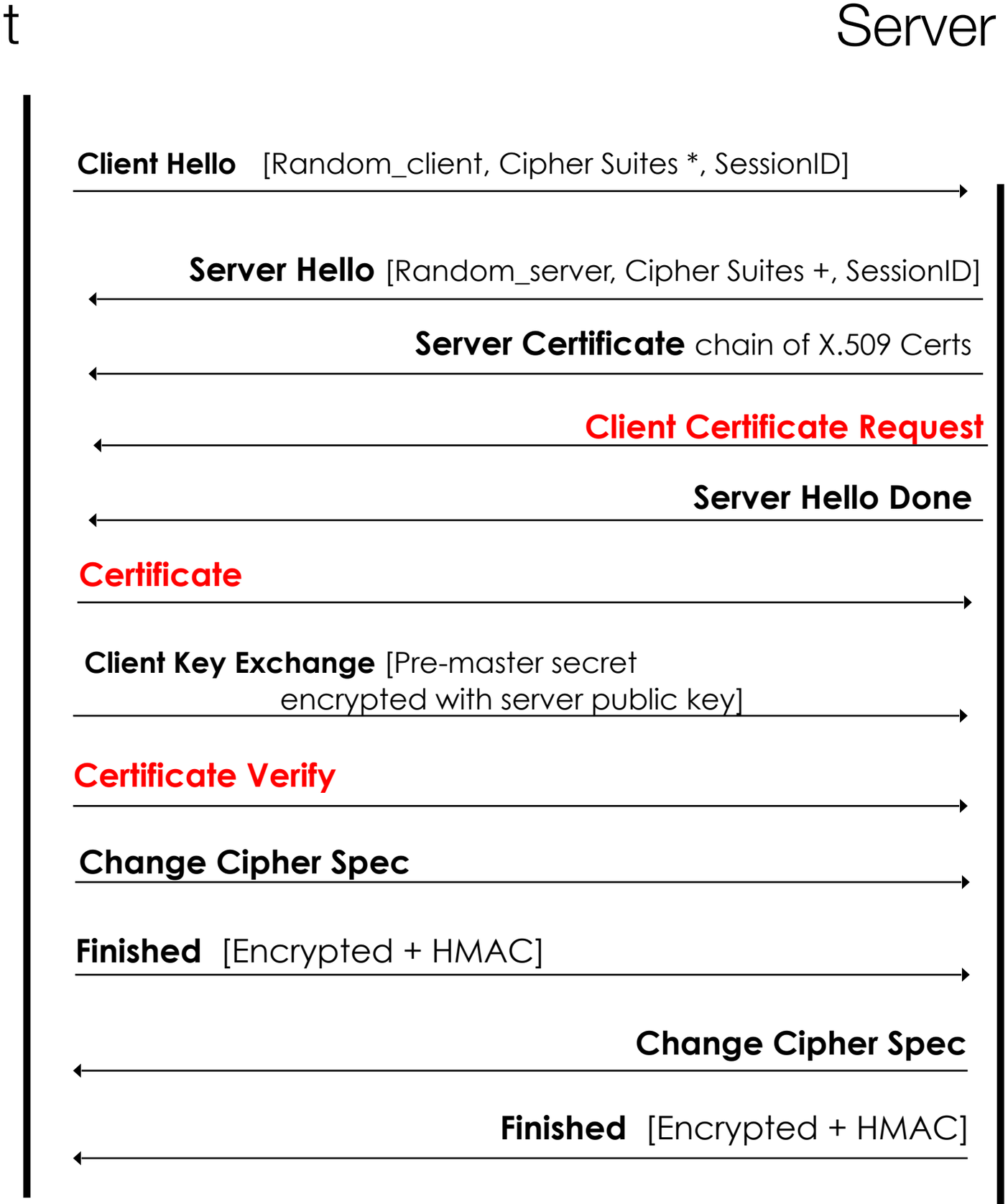
- The Finished message is the first one that is encrypted using the master secret
- The Finished message also includes an HMAC of *part* of the previously exchanged messages to ensure nobody tampered with the handshake

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/

RSA Key Exchange Method

Client Authentication



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

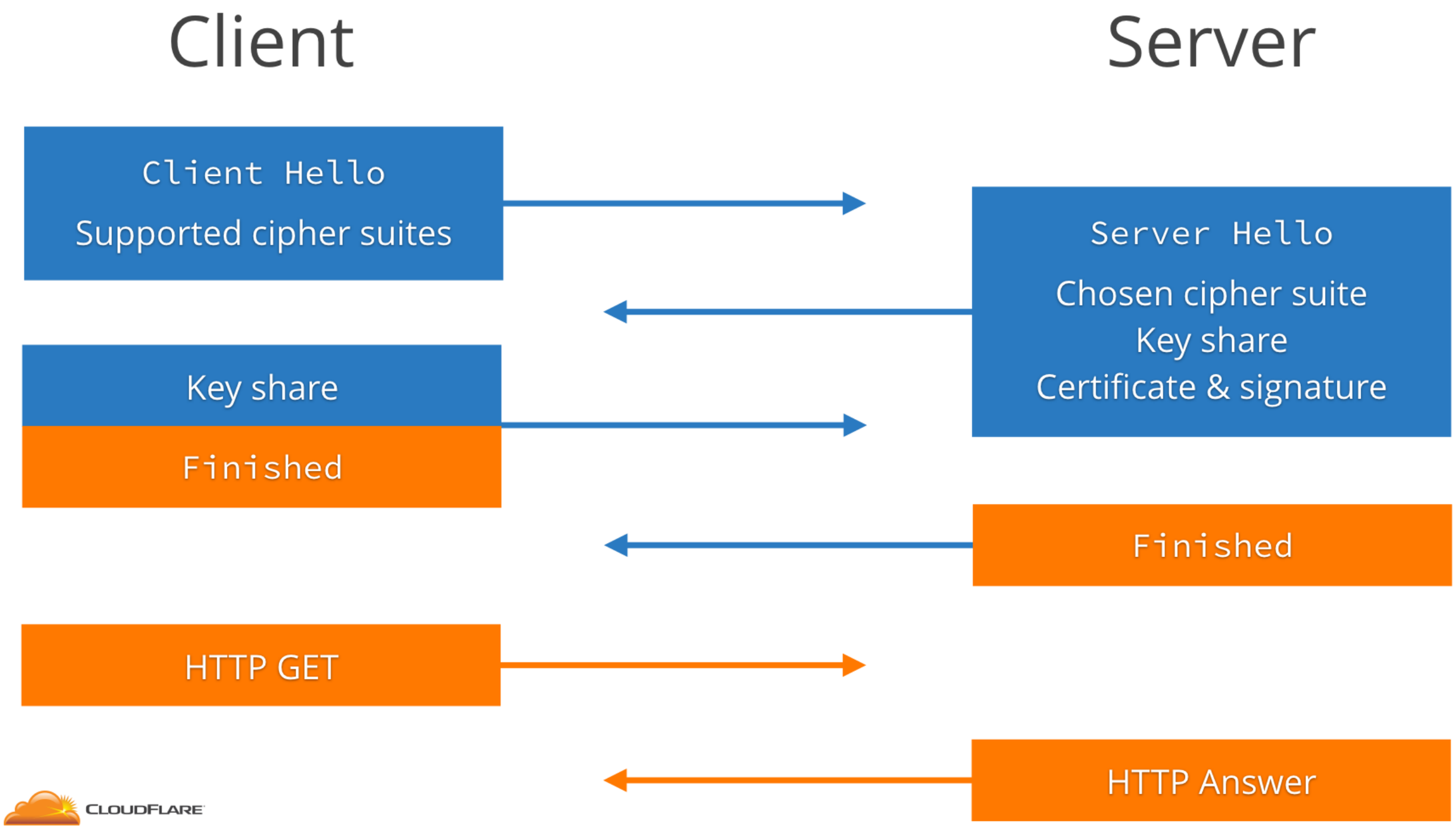
Perfect Forward Secrecy

- 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 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)

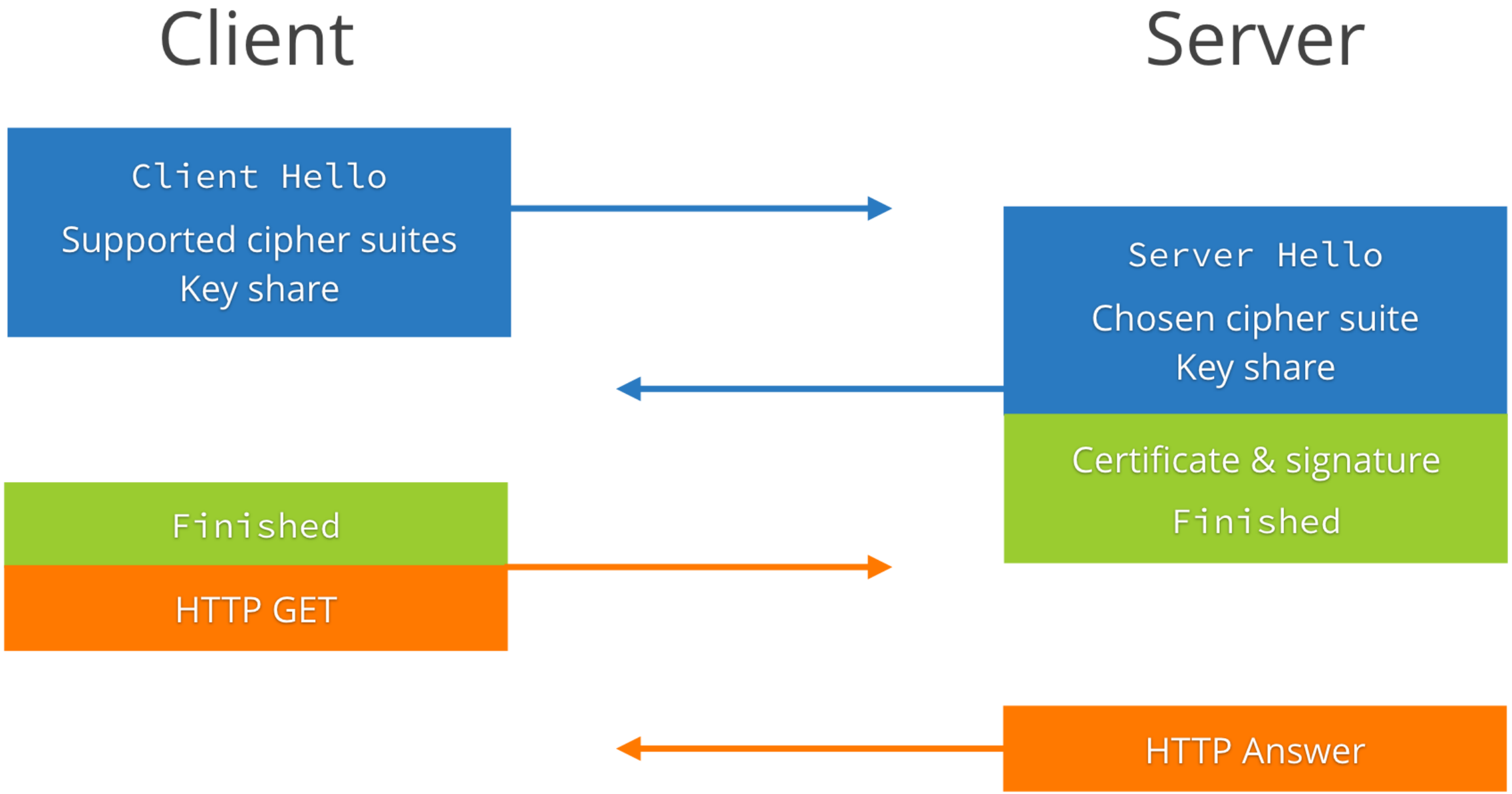
TLS 1.3

- Improvements
    - Reduced round trips in the handshake
    - Certificates are encrypted
    - Quick session resumption
    - Signature covers the *entire* handshake
- Resources
    - https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/
    - https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a/

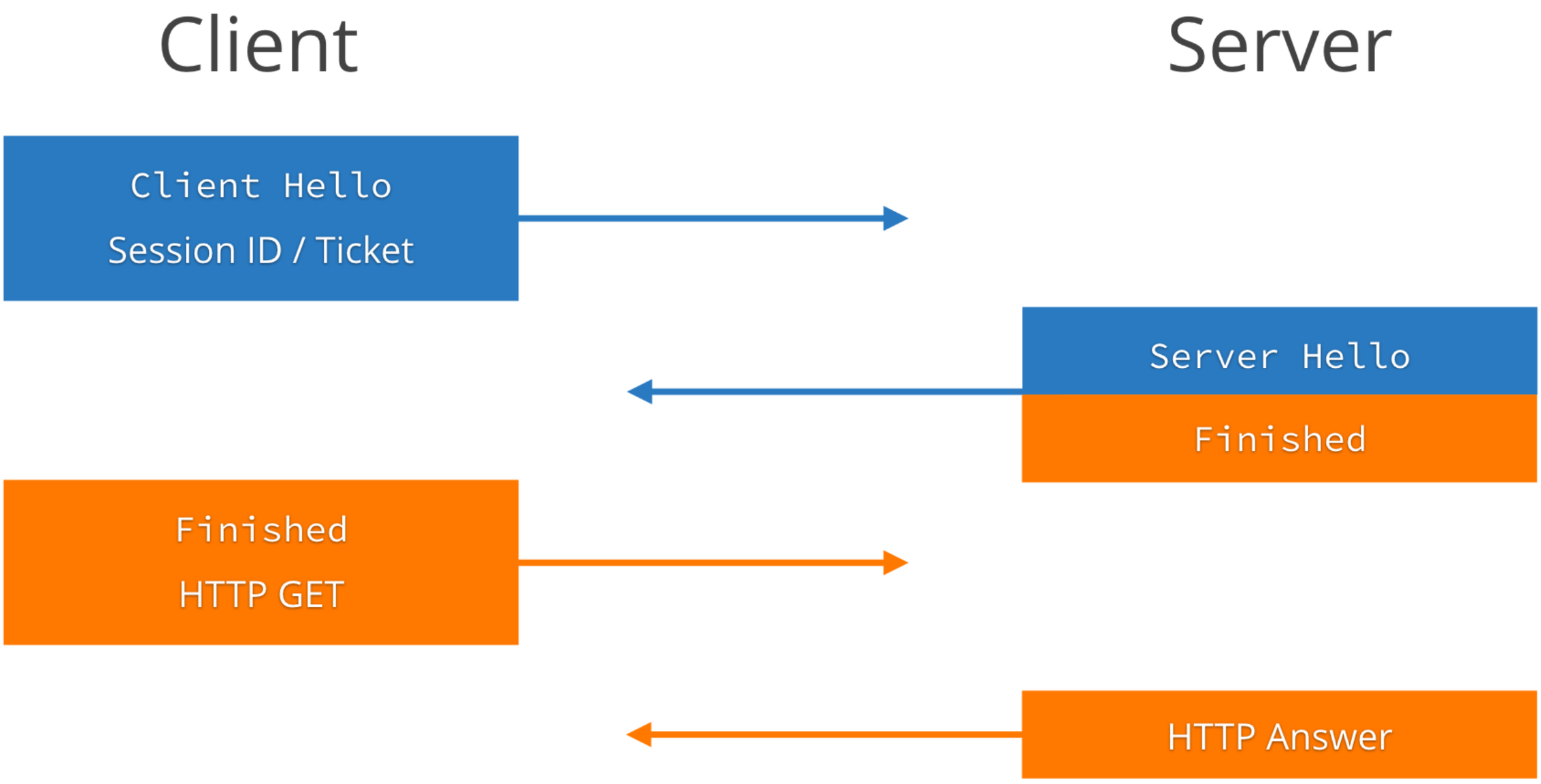
TLS 1.2 (Simplified)



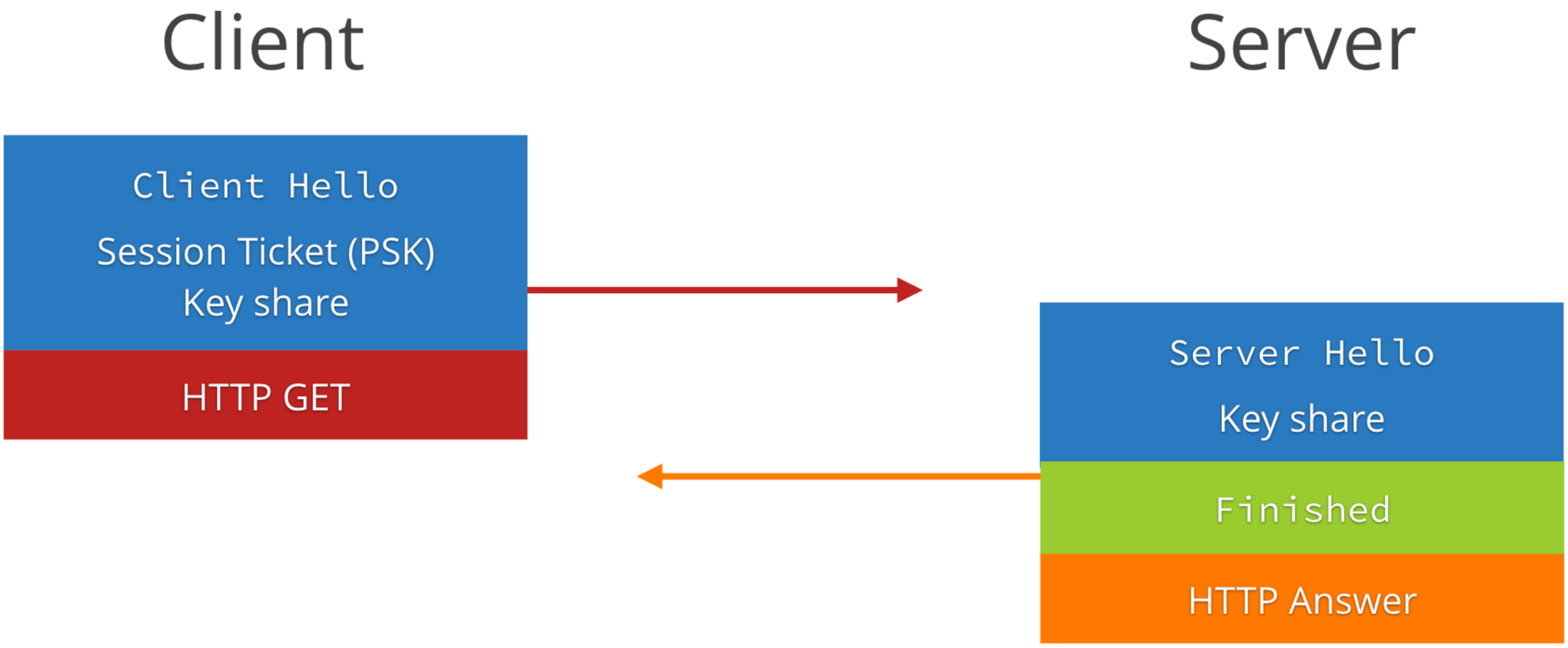
TLS 1.3 (Simplified)



TLS 1.2 Resumption



TLS 1.3 Resumption (0-RTT)



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