PennState

CSE 543: Computer Security

Module: Network Security

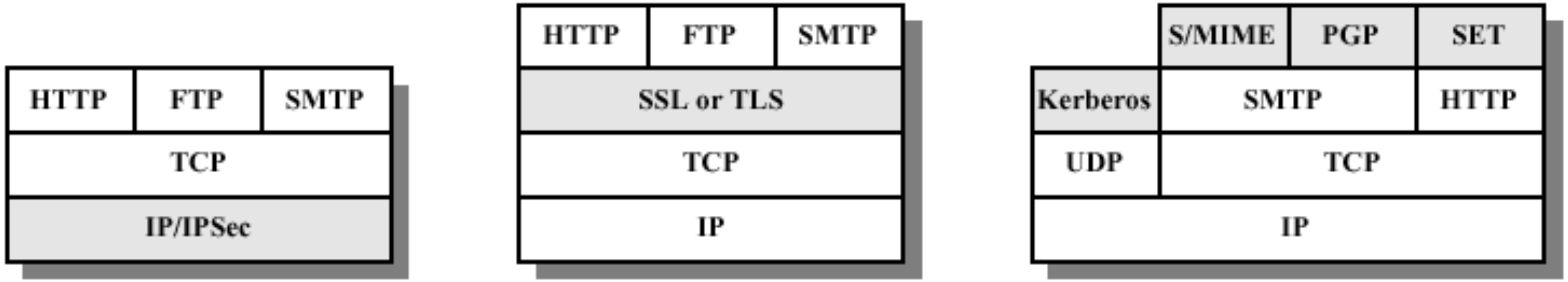
Network Protocols

Prof. Syed Rafiul Hussain
Department of Computer Science and Engineering
The Pennsylvania State University

Communication Security

- Want to establish a secure channel to remote hosts over an untrusted network
  - Users - when logging in to a remote host
  - Applications - when communicating across network
  - Hosts - when logically part of the same isolated network
- The communication service must …
  - Authenticate the end-points (each other)
  - Negotiate what security is necessary (and how achieved)
  - Establish a secure channel (e.g., key distribution/agreement)
  - Process the traffic between the end points
- Also known as *communications security*.

Security Facilities in TCP/IP Stack



(a) Network Level (b) Transport Level (c) Application Level

Transport Security

PennState

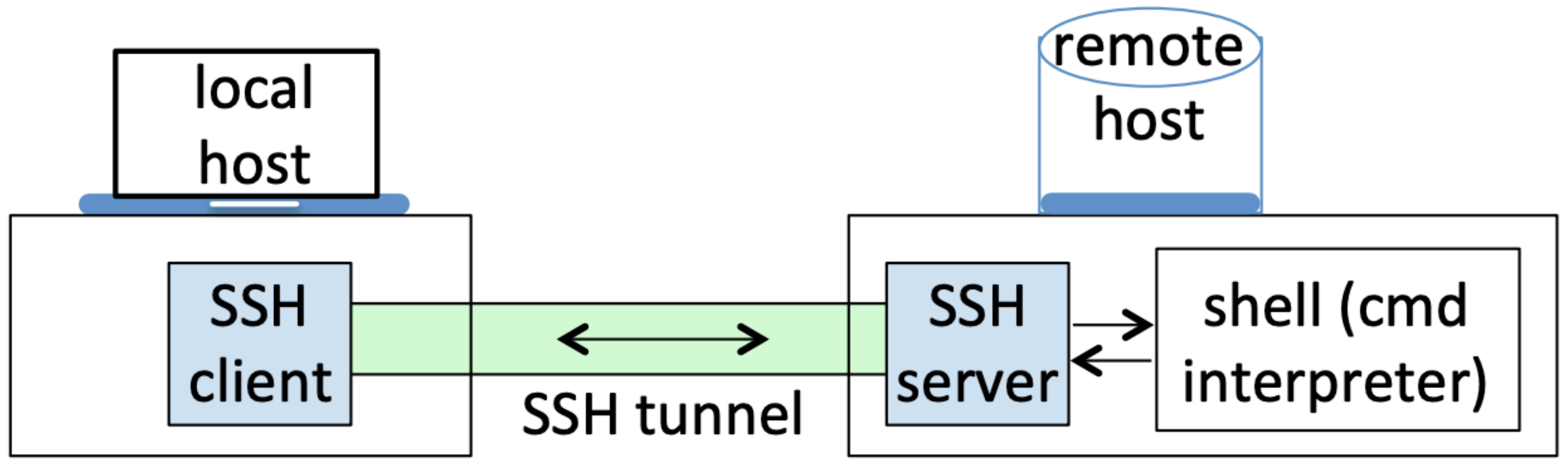
- A host wants to establish a secure channel to remote hosts over an untrusted network
  - Not Login – end-users may not even be aware that protections in place (transparent)
  - Remote hosts may be internal or external
- The protection service must …
  - Authenticate the end-points (each other)
  - Negotiate what security is necessary (and how achieved)
  - Establish a secure channel (e.g., key distribution/agreement)
  - Process the traffic between the end points
- Also known as *communications security*.

Users' Communications Security

- Login to a host over an untrusted network
  - Using unauthenticated login - telnet, rsh - up to this point
- Problems
  - How does user authenticate host?
  - How does host authenticate user?

SSH (Secure Shell)

- Secure communication protocol…
  - Between user's client and remote machine (server)
  - Used to implement remote login
  - Runs on any transport layer (TCP/IP)
- Setup
  - Authentication agent on client
    - To produce and process messages on behalf of user
  - SSH Server
    - To handle user logins to that host
    - Forward X and TCP communications
- Remote machine use approximates local machine



Remote shell through an SSH tunnel, providing authenticated encryption.

SSH Protocol

- How to authenticate server-user and user-server?
  - Users lack public keys
  - But, servers may hold login passwords of users

SSH Protocol

- How to authenticate server-user and user-server?
  - Users lack public keys
  - But, servers may hold login passwords of users
- How to establish a secure channel?
  - Between the client and server
  - For remote processing of commands

SSH Protocol

- (1) Client opens connection to server
- (2) Server responds with its host key and server key
  - Public keys identifying server and enabling communication
- (3) Client generates random number and encrypts with host and server keys
- (4) Server extracts random number (key) and can use
  - Server is authenticated
- (5) Server authenticates user
  - Password and RSA authentication
- (6) Preparatory phase
  - To setup TCP/IP, X11 forwarding, etc.
- (7) Interactive session phase

SSH Protocol

- How to authenticate server-user and user-server?
  - Users lack public keys
  - But, servers may hold login passwords of users

- **Answer**:

SSH Protocol

- How to authenticate server-user and user-server?
  - Users lack public keys
  - But, servers may hold login passwords of users

- **Answer**: Server public keys (host and server) and user passwords

- How are we sure that these are the legitimate public keys for the server?

SSH Protocol

- How to authenticate server-user and user-server?
  - Users lack public keys
  - But, servers may hold login passwords of users
- How to establish a secure channel?
  - Between the client and server
  - For remote processing of commands

- **Answer**:

SSH Protocol

- How to authenticate server-user and user-server?
  - Users lack public keys
  - But, servers may hold login passwords of users
- How to establish a secure channel?
  - Between the client and server
  - For remote processing of commands

- **Answer**: Client chooses key

- How does client know what kind of key to pick?

SSHv2 Protocol

- A number of improvements were made to the SSHv2 protocol (see Section 5)
  - Stronger use of crypto - better algorithms
  - Performance - 1.5 round trips on average
  - Prevent eavesdropping - encrypt all SSH traffic
  - Prevent IP spoofing - always validates server identity
  - Prevent hijacking - integrity checking using HMAC
- Not backwards compatible with SSHv1

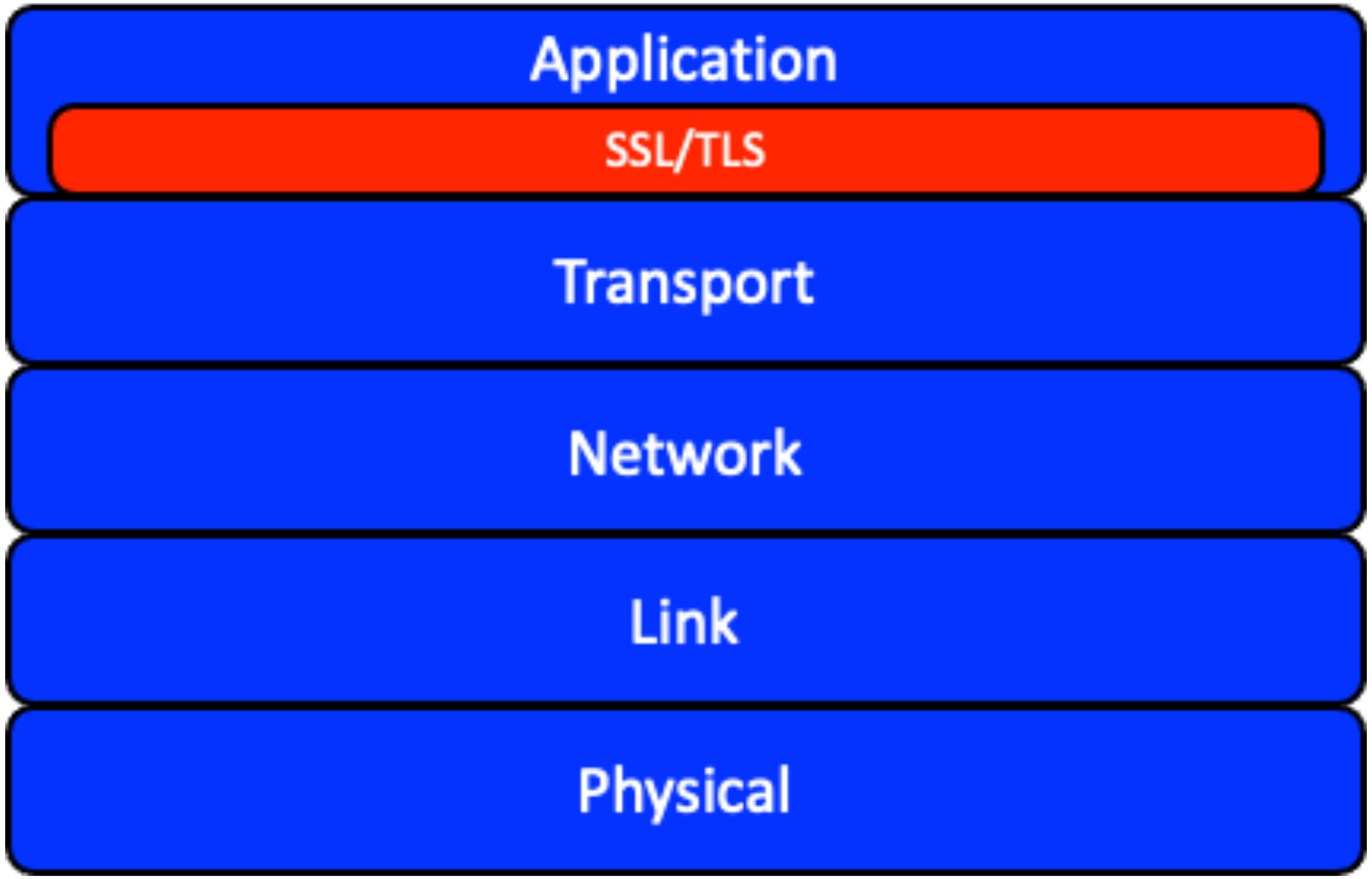
Application Comm Security

- Applications may want to construct secure communication channels transparently to users
  - How can they do that?

SSL/TLS

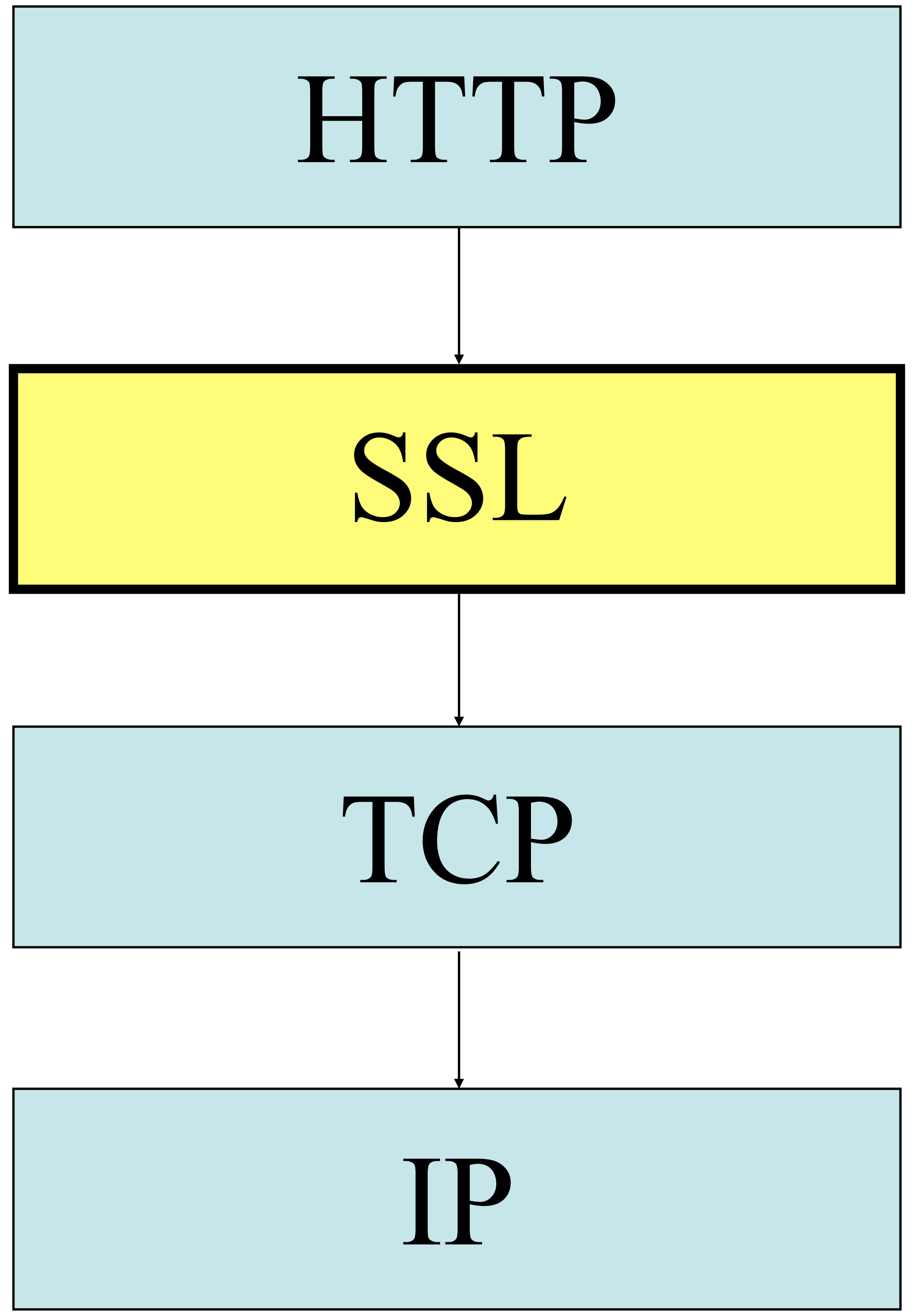
- Secure Socket Layer (SSL)
  - v2 Developed by Netscape Navigator in 1995
  - v3 released in 1996
- Transport Layer Security (TLS)
  - Released as RFC in 1999
  - Attempt to standardize the protocol
- Basic idea: A program can replace socket creation with a "secure socket" to get authentication, confidentiality and integrity
- HTTPS = HTTP + SSL/TLS
-

Network Stack Revisited



Application (Web) Security: SSL

- Secure socket Layer (SSL/TLS)
  - Used to authenticate servers
    - Uses certificates, “root” CAs
  - **Can** authenticate clients
  - Inclusive security protocol
  - Security at the socket layer
  - Transport Layer Security (TLS)
    - Provides
      - authentication
      - confidentiality
      - integrity

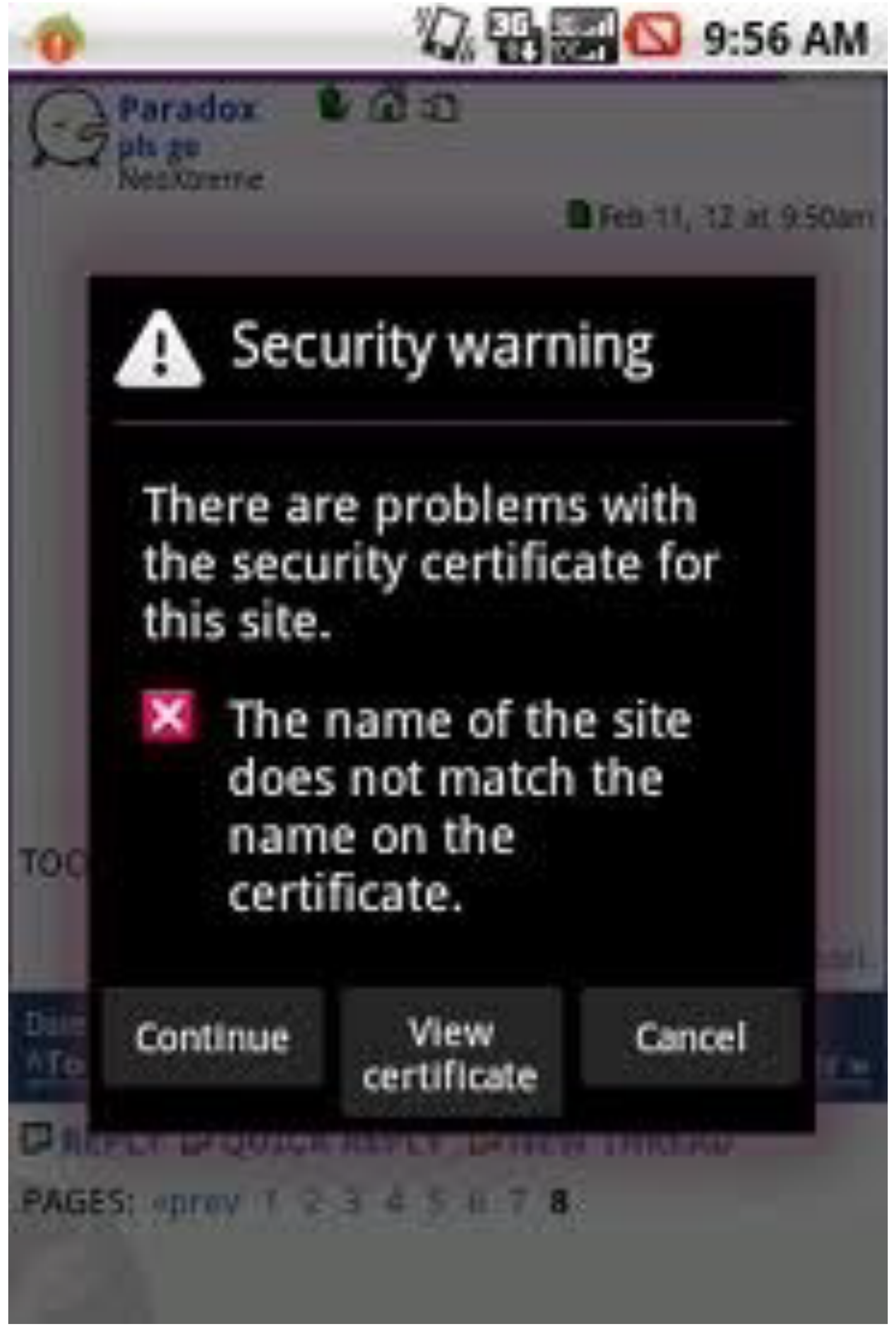


Where is this useful

- Online commerce
  - Varying risk between client and server (customer and retailer)
- Web services
  - Secure password authentications!
- Session establishment for internet applications (e.g. VoIP)
- VPN connections

Security Guarantees

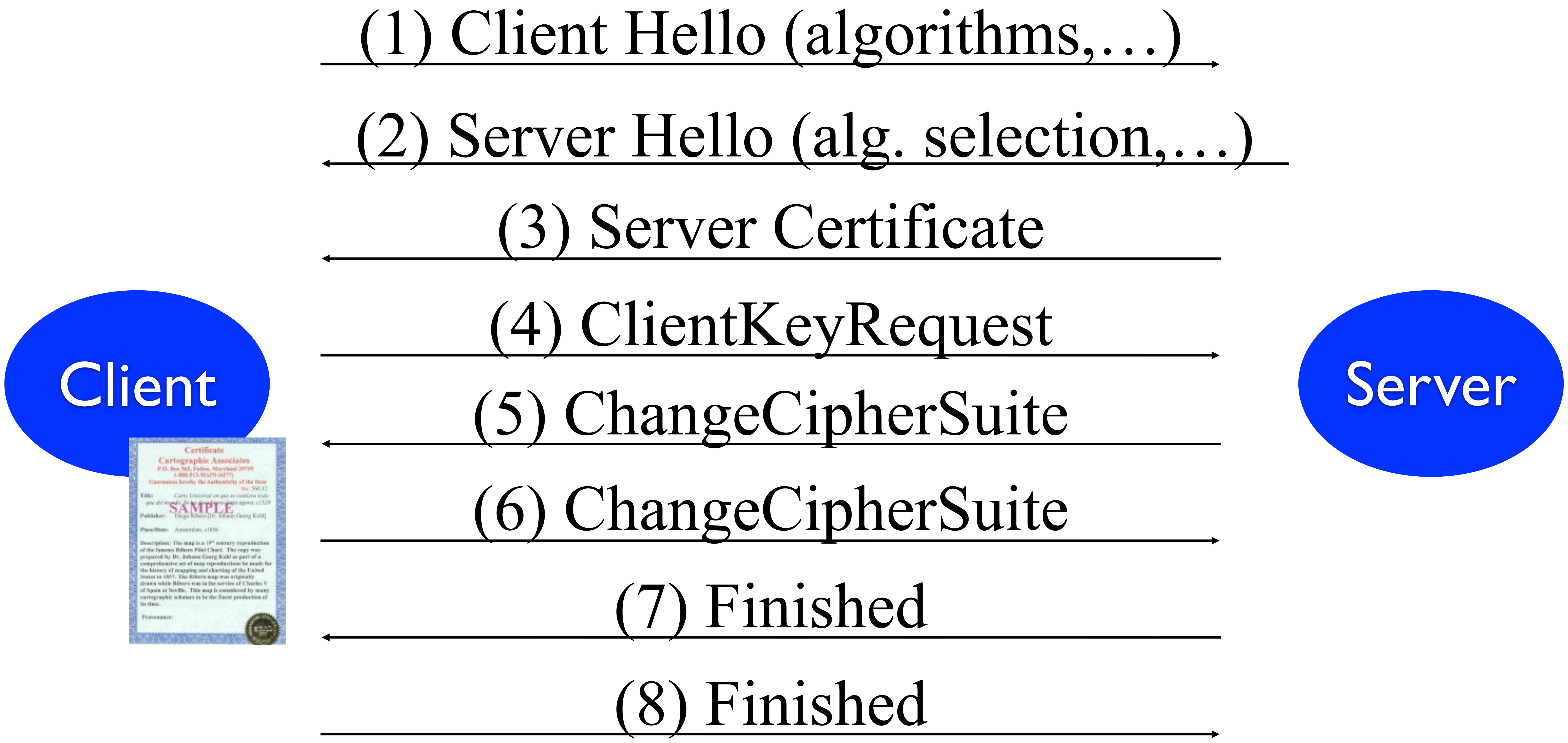
- Server authentication
  - Requires certificate infrastructure
  - Can also provide client authentication, rarely used
- Session key establishment
  - Confidentiality, Authentication, Integrity
- Built-in functionality
  - Integrated into browsers
  - Browsers include set of trusted CA root certificates



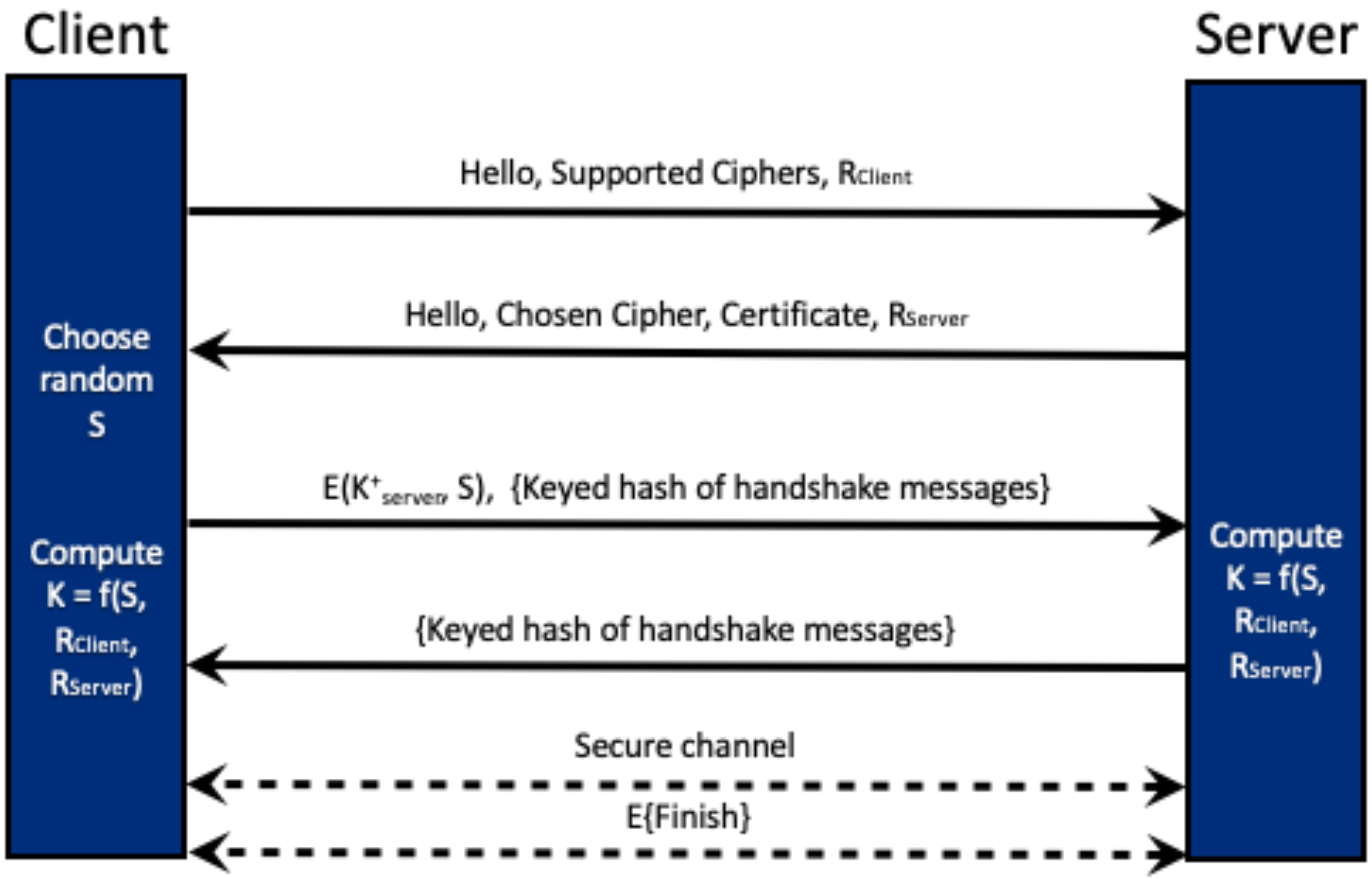
A simple SSL protocol description

- Client initiates connection
- Server chooses security parameters
- Client generates keying material
- Server authenticates and replicates keying material
- Client and server communicate

SSL Handshake



Simple SSL


Client
Server
Hello, Supported Ciphers, RClient
Hello, Chosen Cipher, Certificate, RServer
Choose random S
E(K+server, S), {Keyed hash of handshake messages}
Compute K = f(S, RClient, RServer)
{Keyed hash of handshake messages}
Compute K = f(S, RClient, RServer)
Secure channel
E{Finish}

Simplified Protocol Detail

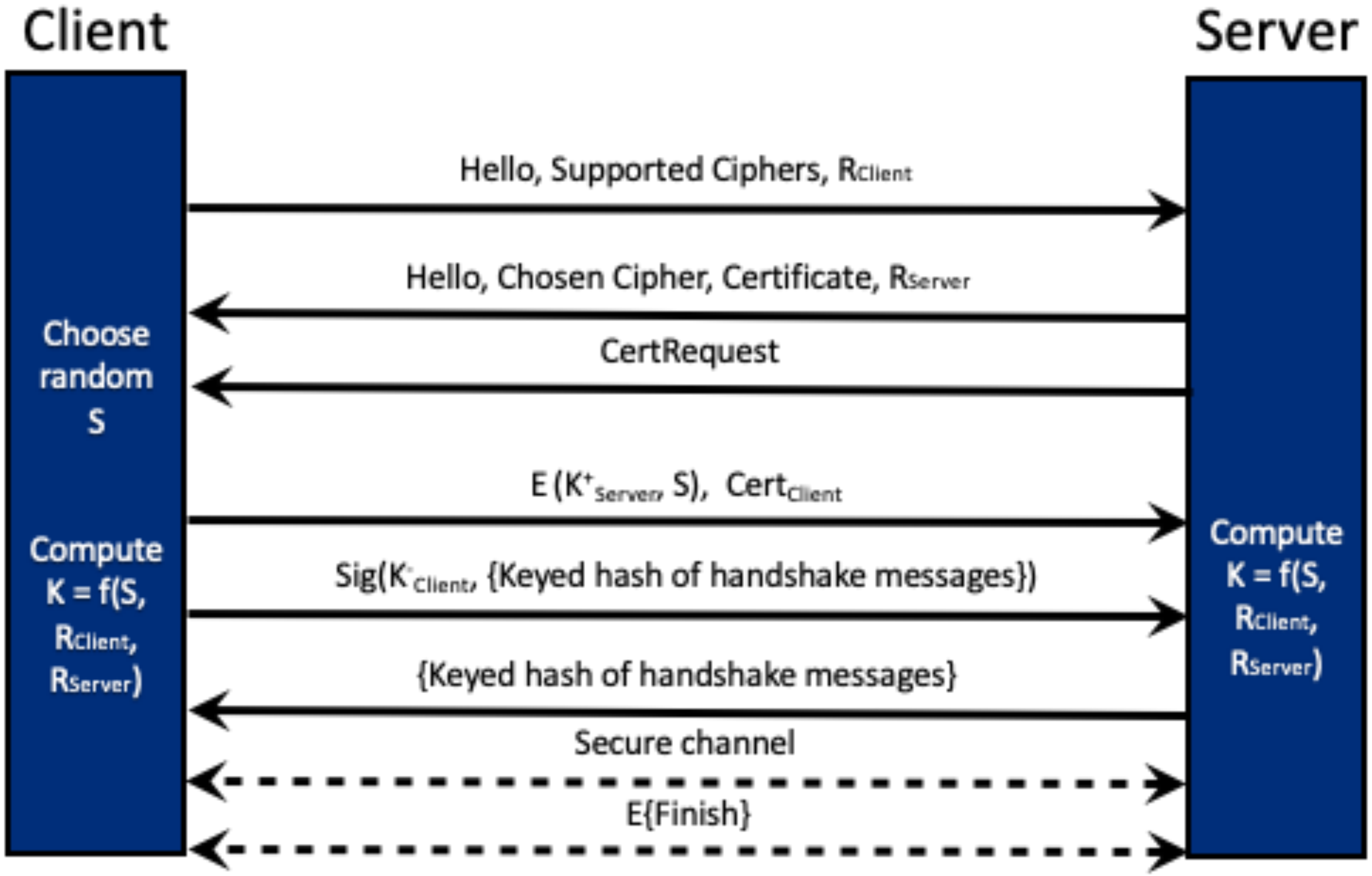
Participants: Alice/A (client) and Bob/B (server)
Crypto Elements : Random R, Certificate C, k_i^+ Public Key (of i)
Crypto Functions : Hash function $H(x)$, Encryption $E(k,d)$, Decryption $D(k,d)$, Keyed MAC $HMAC(k,d)$

1.	Alice $\rightarrow$ Bob	R_A
2.	Bob $\rightarrow$ Alice	R_B, C_B
	Alice	pick pre-master secret S
	Alice	calculate master secret $K = H(S, R_A, R_B)$
3.	Alice $\rightarrow$ Bob	$E(k_B^+, S), HMAC(K,' CLNT' + [\#1, \#2])$
	Bob	recover pre-master secret $S = D(k_B^-, E(k_B^+, S))$
	Bob	calculate master secret $K = H(S, R_A, R_B)$
4.	Bob $\rightarrow$ Alice	$HMAC(K,' SRVR' + [\#1, \#2])$

Note: Alice and Bob : IV Keys, Encryption Keys, and Integrity Keys 6 keys,where each key $k_i = g_i(K, R_A, R_B)$, and g_i is key generator function.

CLNT' = Alice, SRVR' = Bob

Simple SSL (With Client Auth)



The Full Protocol

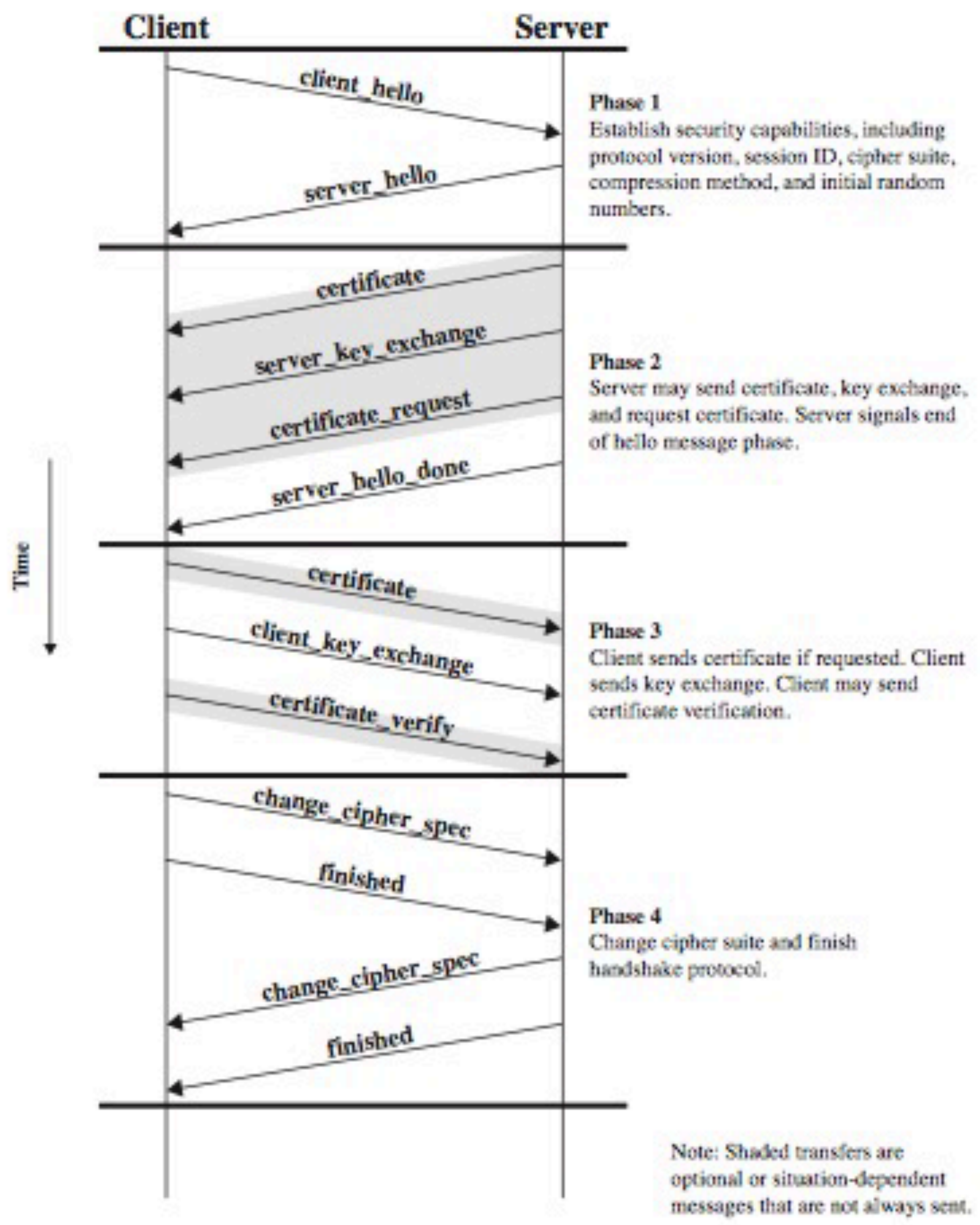


Figure 6.6 Handshake Protocol Action

Phase 1

- Client Hello:
  - Protocol version
  - Cipher suites available
  - Random value R_{Client}
  - Session ID (if re-establishment allowed)
- Server Hello:
  - Protocol version
  - Cipher suite chosen
  - Random value R_{Server}
  - Session ID (if re-establishment allowed)

Cipher Suite

- Includes encryption algorithm, key length, block mode, and integrity checksum algorithm
- ~90 defined cipher suites
- Client gives Server a list of supported cipher suites
  - Server makes final choice

```
% openssl ciphers -v
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(256) Mac=SHA384
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA384
ECDHE-RSA-AES256-SHA SSLv3 Kx=ECDH Au=RSA Enc=AES(256) Mac=SHA1
ECDHE-ECDSA-AES256-SHA SSLv3 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA1
SRP-DSS-AES-256-CBC-SHA SSLv3 Kx=SRP Au=DSS Enc=AES(256) Mac=SHA1
SRP-RSA-AES-256-CBC-SHA SSLv3 Kx=SRP Au=RSA Enc=AES(256) Mac=SHA1
SRP-AES-256-CBC-SHA SSLv3 Kx=SRP Au=SRP Enc=AES(256) Mac=SHA1
DHE-DSS-AES256-GCM-SHA384 TLSv1.2 Kx=DH Au=DSS Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH Au=RSA Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-SHA256 TLSv1.2 Kx=DH Au=RSA Enc=AES(256) Mac=SHA256
DHE-DSS-AES256-SHA256 TLSv1.2 Kx=DH Au=DSS Enc=AES(256) Mac=SHA256
DHE-RSA-AES256-SHA SSLv3 Kx=DH Au=RSA Enc=AES(256) Mac=SHA1
DHE-DSS-AES256-SHA SSLv3 Kx=DH Au=DSS Enc=AES(256) Mac=SHA1
DHE-RSA-CAMELLIA256-SHA SSLv3 Kx=DH Au=RSA Enc=Camellia(256) Mac=SHA1
DHE-DSS-CAMELLIA256-SHA SSLv3 Kx=DH Au=DSS Enc=Camellia(256) Mac=SHA1
ECDH-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH/RSA Au=ECDH Enc=AESGCM(256) Mac=AEAD
ECDH-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH/ECDSA Au=ECDH Enc=AESGCM(256) Mac=AEAD
ECDH-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH/RSA Au=ECDH Enc=AES(256) Mac=SHA384
ECDH-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH/ECDSA Au=ECDH Enc=AES(256) Mac=SHA384
ECDH-RSA-AES256-SHA SSLv3 Kx=ECDH/RSA Au=ECDH Enc=AES(256) Mac=SHA1
ECDH-ECDSA-AES256-SHA SSLv3 Kx=ECDH/ECDSA Au=ECDH Enc=AES(256) Mac=SHA1
AES256-GCM-SHA384 TLSv1.2 Kx=RSA Au=RSA Enc=AESGCM(256) Mac=AEAD
AES256-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AES(256) Mac=SHA256
AES256-SHA SSLv3 Kx=RSA Au=RSA Enc=AES(256) Mac=SHA1
CAMELLIA256-SHA SSLv3 Kx=RSA Au=RSA Enc=Camellia(256) Mac=SHA1
PSK-AES256-CBC-SHA SSLv3 Kx=PSK Au=PSK Enc=AES(256) Mac=SHA1
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(128) Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(128) Mac=SHA256
```

Phase 2

- Server authentication
  - Public Key Certificate
  - Optional steps:
- Key exchange message
- Request for client certificate
  - Server hello done
- Why is this necessary?

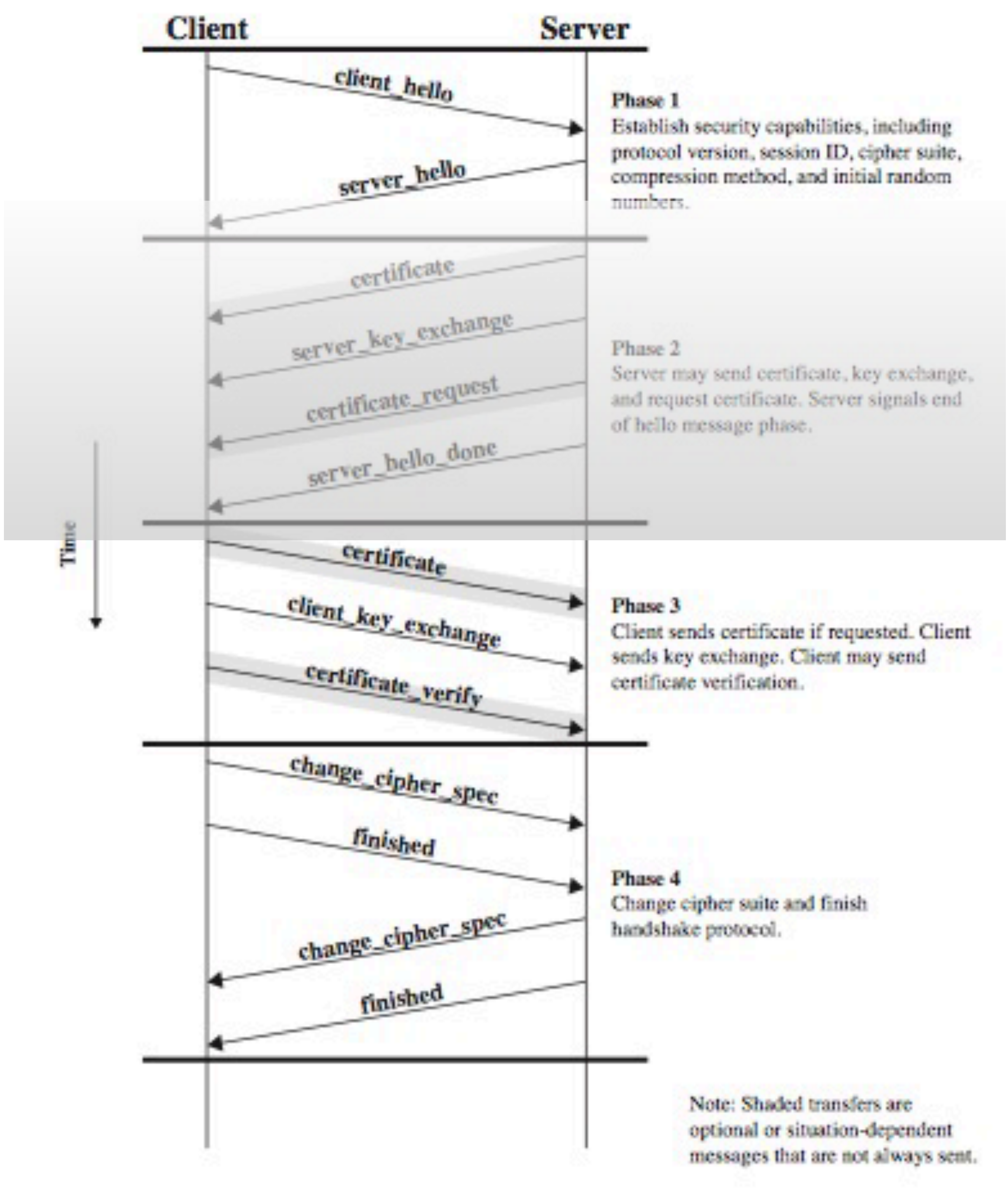


Figure 6.6 Handshake Protocol Action

Phase 3

- Client key exchange
  - Client generates secret S, encrypts using server's public key
  - Optional messages:
    - Client certificate
    - Certificate verification

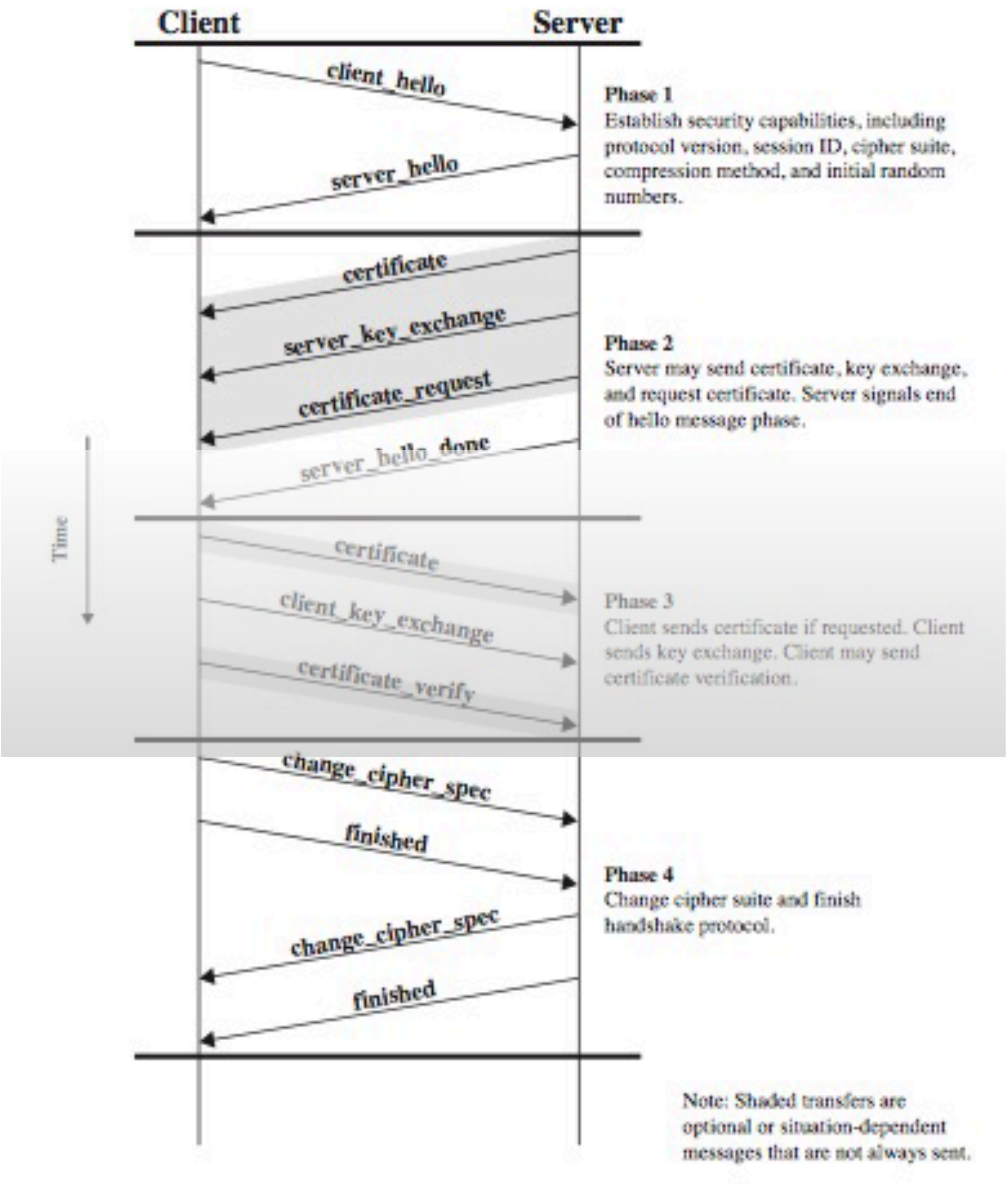


Figure 6.6 Handshake Protocol Action

Key Exchange Methods

- RSA (server must have a certificate)
- Fixed Diffie-Hellman
  - Server provides DH public parameters in a certificate
  - Client responds with DH public key in a certificate or key exchange message
- Ephemeral Diffie-Hellman
  - DH public keys are exchanged, signed by RSA key
- Anonymous Diffie-Hellman
  - DH parameters with no authentication

Phase4

- Generate the primary secret
  - f(S, R_{Client}, R_{Server})
- Client finish:
  - change_cipher_spec
  - finished
- Server finish:
  - change_cipher_spec
  - finished

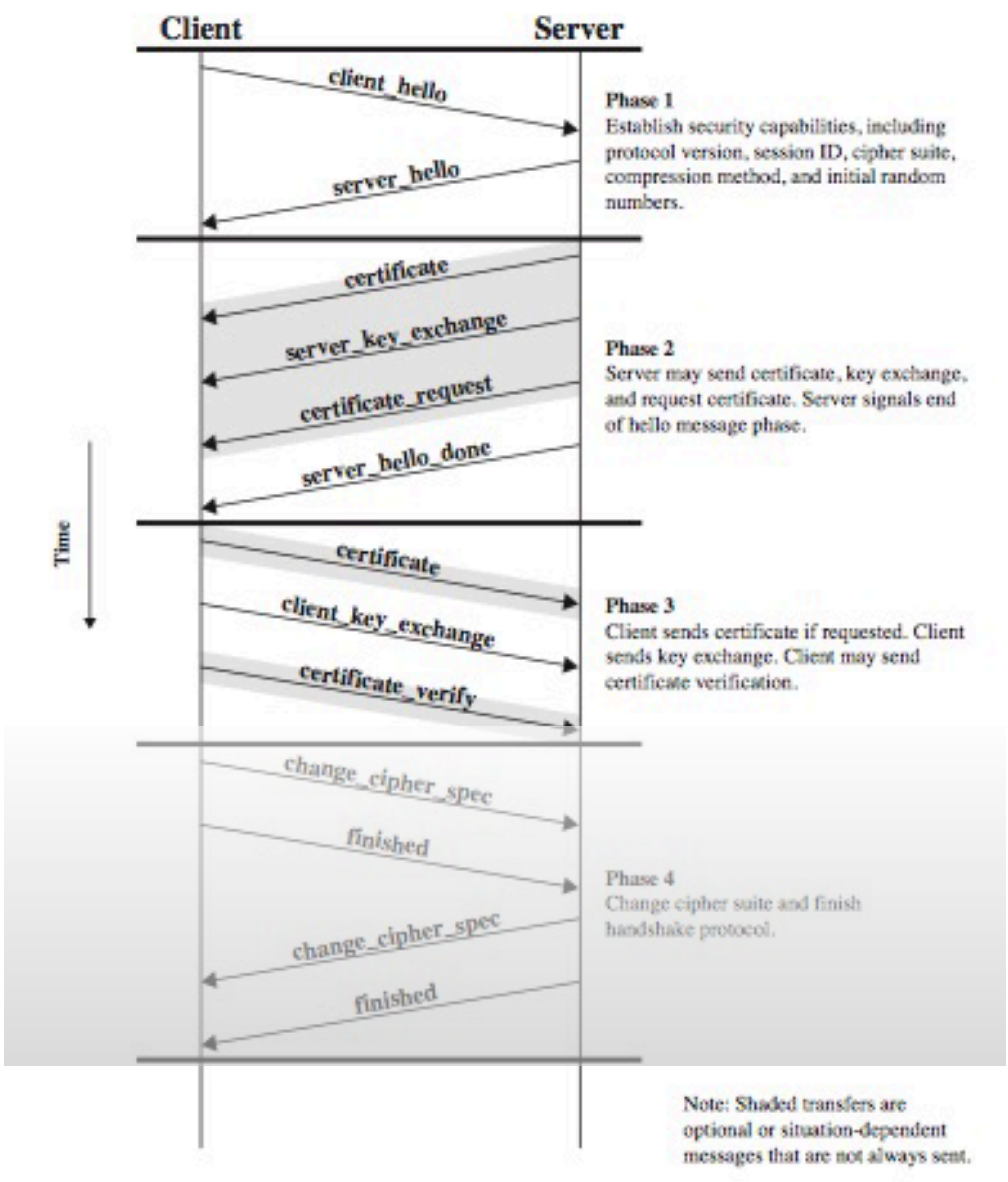


Figure 6.6 Handshake Protocol Action

Cryptographic Parameters

- Generated from
  - the primary secret K
  - R_{Client}
  - R_{Server}
- Six values to be generated
  - client authentication and encryption keys
  - server authentication and encryption keys
  - client encryption IV
  - server encryption IV
- Generator functions: $k_i = g_i(K, R_{Client}, R_{Server})$

SSL Tradeoffs

- Pros
  - Server authentication*
  - GUI clues for users
  - Built into every browser
  - Easy to configure on the server
  - Protocol has been analyzed like crazy
- Cons
  - Users don't check certificates
  - Too easy to obtain certificates
  - Too many roots in the browsers
  - Some settings are terrible

Cipher Downgrade

- SSLv2 did not authenticate the client/server hello
- An active adversary could select the cipher suites supported by Alice.
- Select a weak cipher (e.g., the null cipher) as the only supported encryption schemes.
- Fixed in SSLv3
-

Truncation Attack

- SSL v2 did not authenticate the end of session
  - Used TCP FIN for end-of-data
- Allows attacker to keep a login session alive
  - E.g., Causes Gmail and Hotmail to display a page that informs the user they have successfully logged out
  - Big impact for shared computers (e.g., Internet café)
- Fixed in SSLv3 by including end-of-data in the SSL protocol

Why Security Indicators Are Meaningless

PennState

- CA compromise leads to creation of unauthorized certificate
- Server misconfiguration uses a vulnerable cipher suite (like NULL encryption)
- Server attacked to steal private keys for later use
- Web application is vulnerable to CSRF/XSS/SQL Injection
- Malicious code planted on website subverts browser to steal session tokens or authentication information
- International Domain Name (IDN) homograph attacks

- Moral: TLS is a bare minimum to ensure security!

Handshake cost

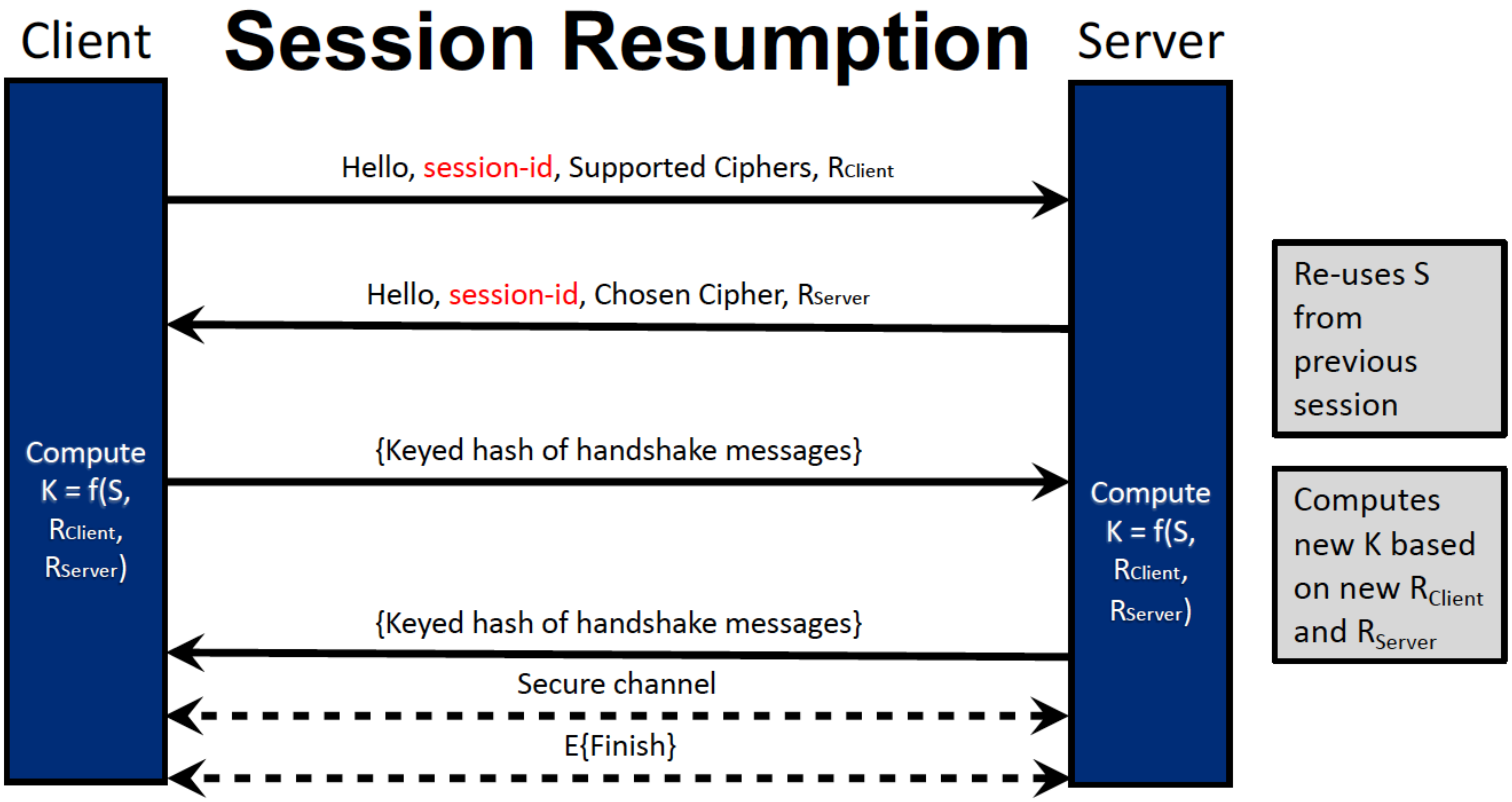
- Per-session master secret derived using expensive public key crypto

```
% openssl speed rsa2048 aes-256-cbc
...
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes  16384 bytes
aes-256 cbc     205394.21k   213467.36k   212927.49k   214833.15k   215094.61k   215633.17k
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000497s 0.000015s   2013.1  67704.7
```

Session Resumption

- Allows Client and Server to construct new encryption & integrity keys using previously shared pre-master secret (S)
  - uses session-id to continue SSL session over multiple connections
  - avoids having to repeat public-key crypto operations
- If either Client or Server don't remember pre-master secret key, new handshake is required

Session Resumption

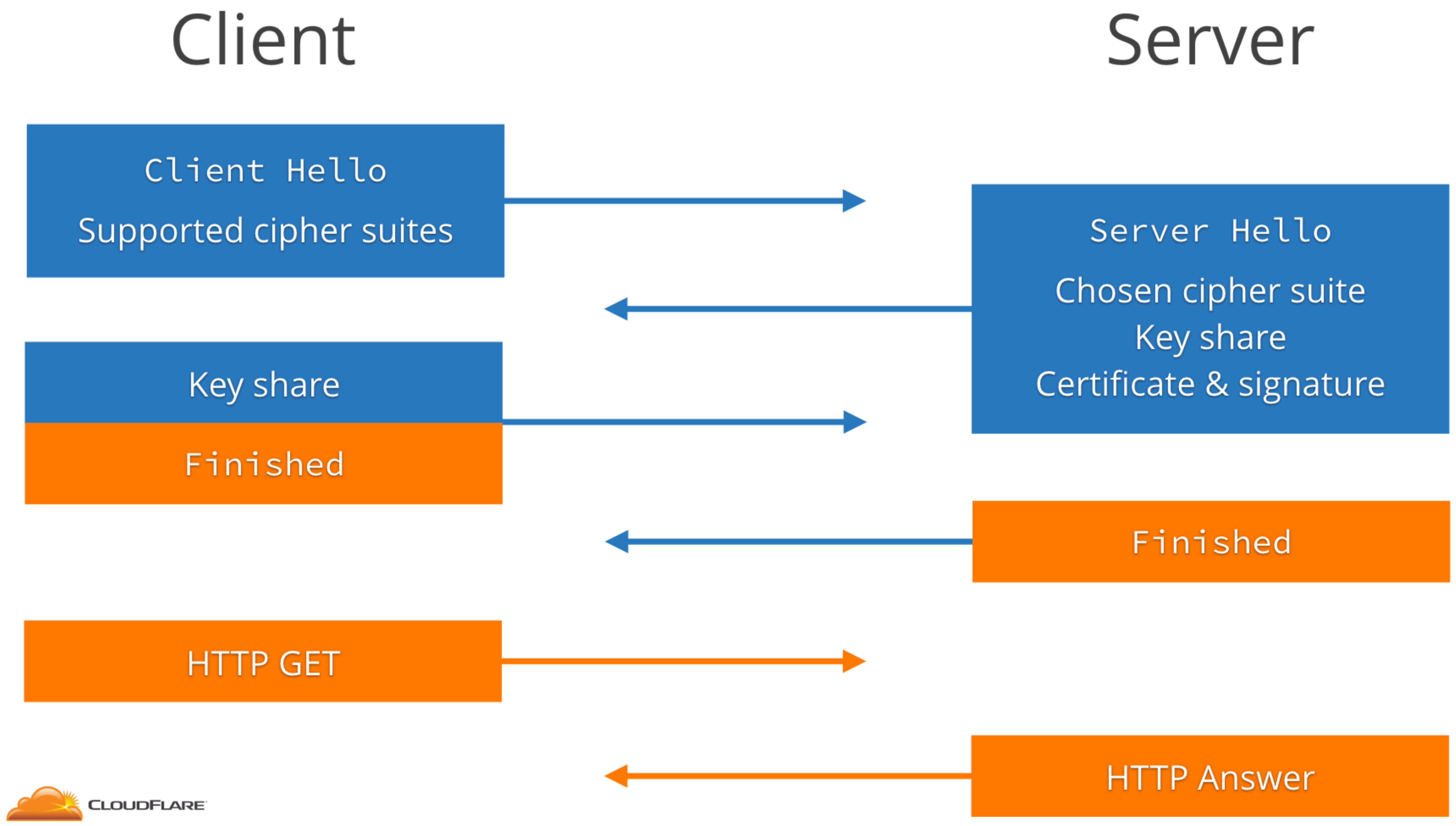


TLS 1.3

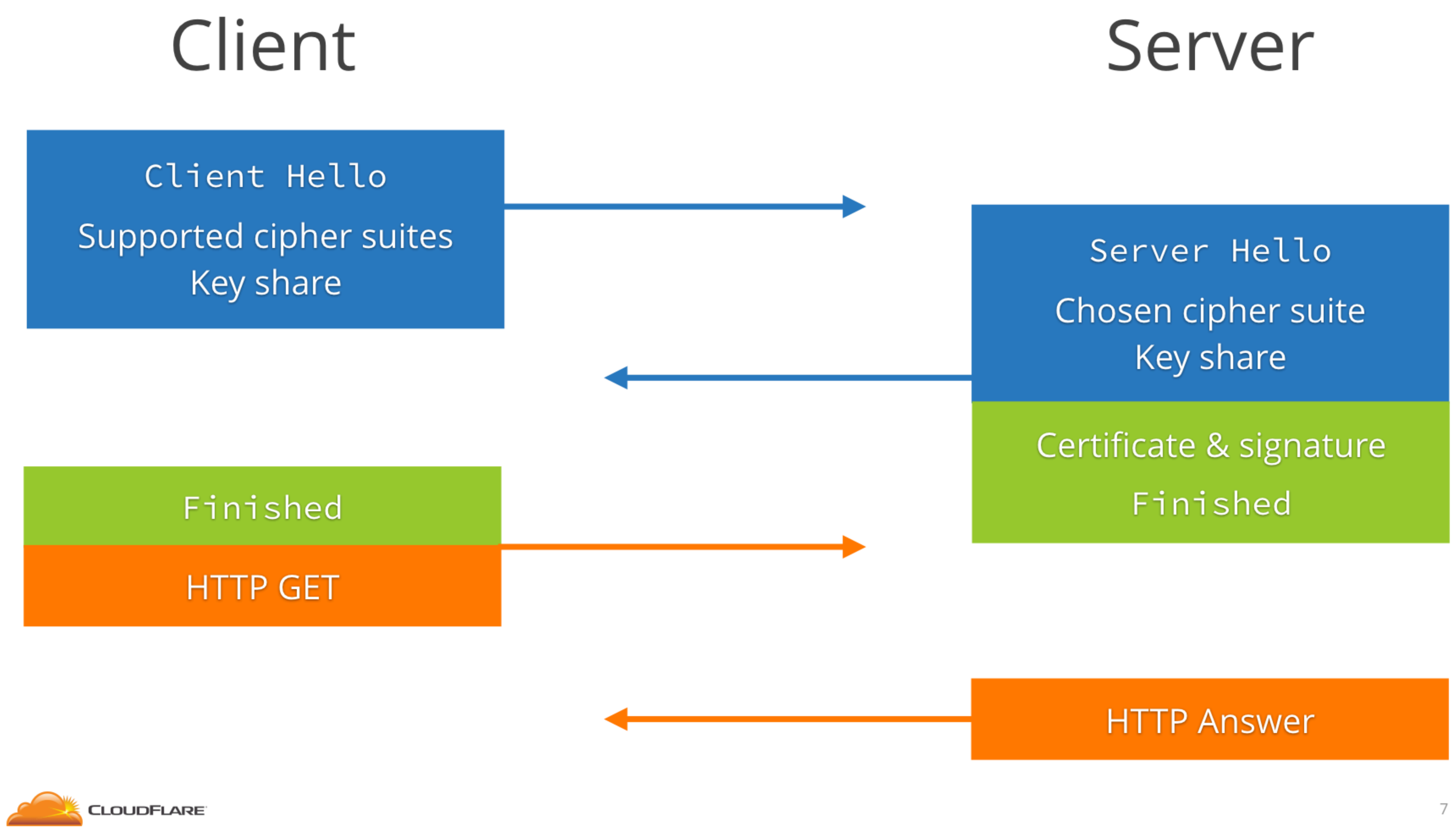
- RFC 8446 Published on August 10, 2018
- Separates key agreement and authentication algorithms from the cipher suites
- Removes some weaker cipher suites
- Mandates perfect forward secrecy using ephemeral keys during DH key agreement
- Supports 1-RTT handshakes and initial support for 0-RTT for
- session resumption

TLS 1.2 vs TLS 1.3

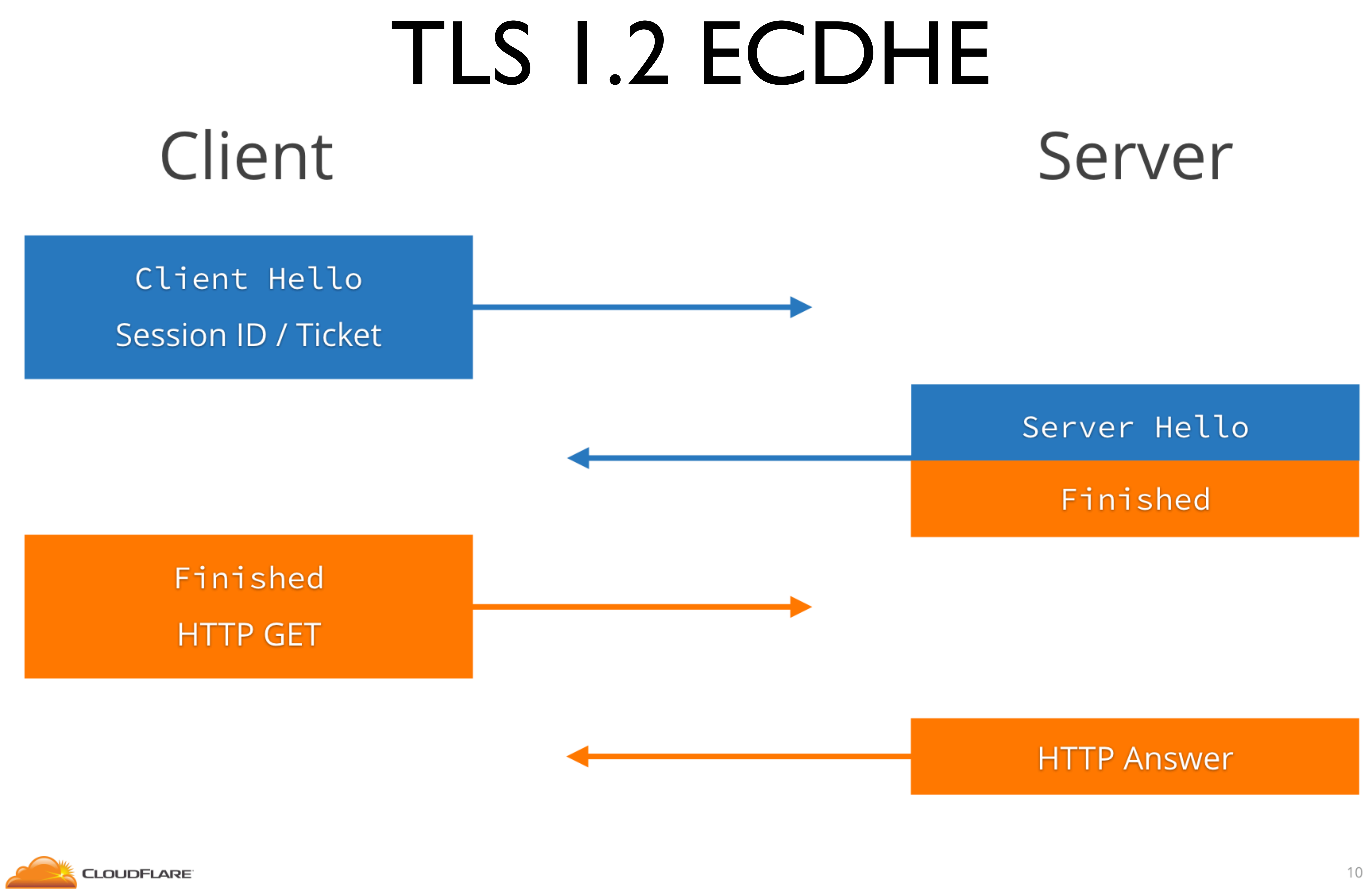
TLS 1.2 ECDHE

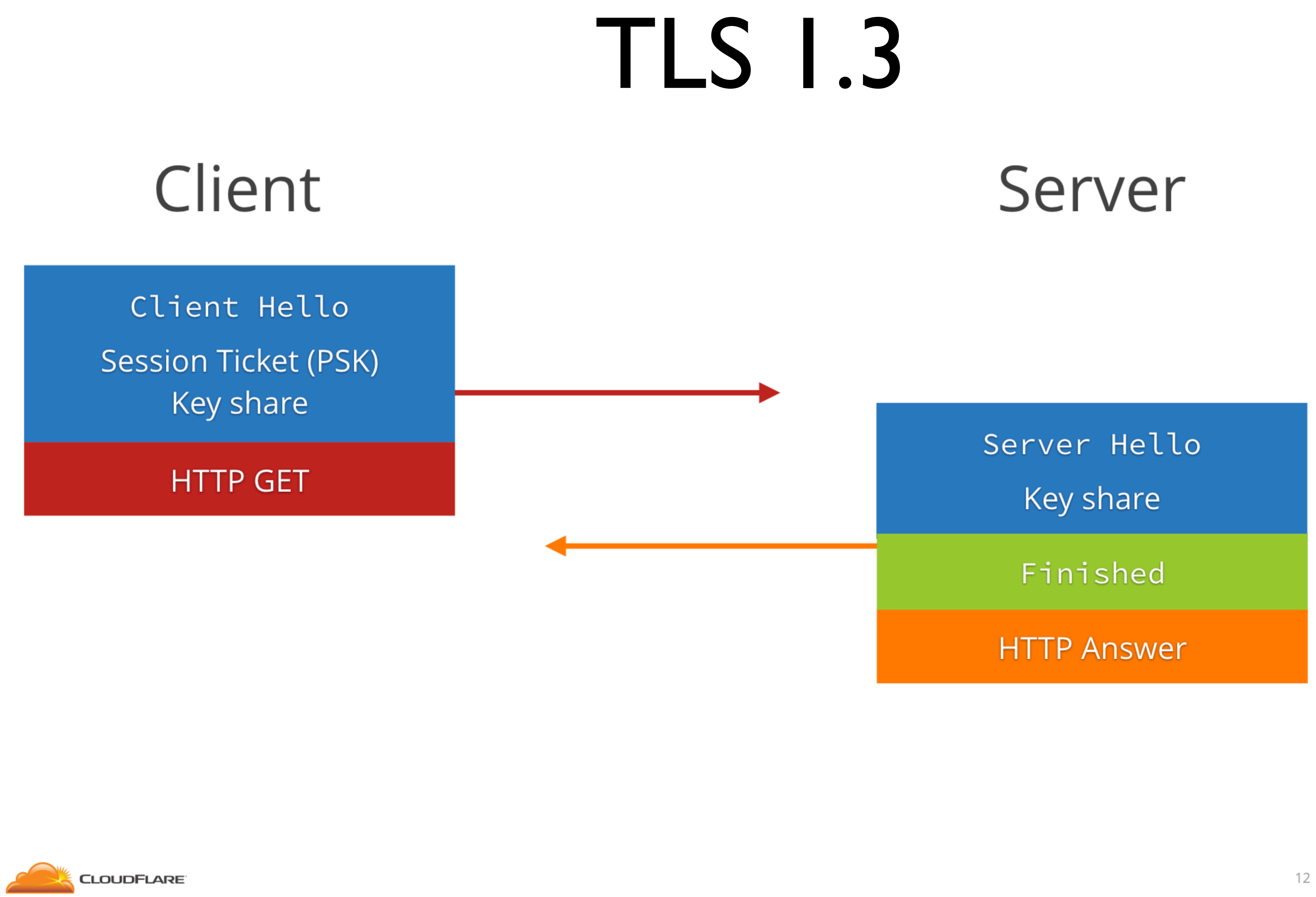


TLS 1.3



TLS 1.2 vs. TLS 1.3 Resumption





Attacks on TLS

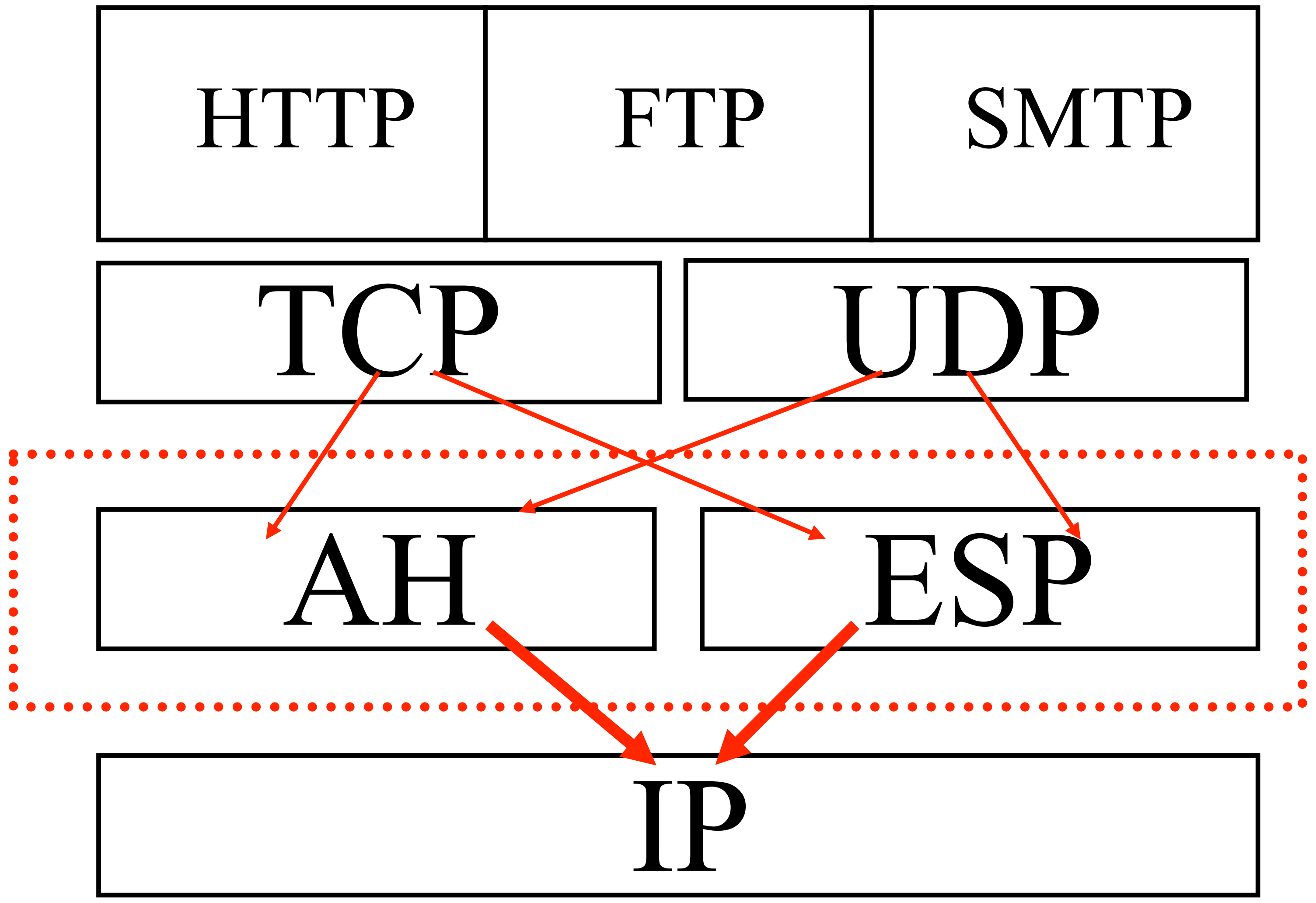
- Downgrade attacks: cause endpoints to use small keys: FREAK (for RSA) and Logjam (for DH)
- Cross-protocol attacks: DROWN attack causes downgrade to SSLv2, which allows weak cipher modes
- BEAST: chosen plaintext attack via a Java applet. Due to quirk in reusing CBC residue as IV for next message
- CRIME and BREACH attacks: recover cookies when data compression is used.
- POODLE: padding oracle attack due to MtE
- Sweet32: affects 64-bit block cipher modes (e.g., 3DES)

IPsec (not IPSec!)

- Host-level protection service
  - IP-layer security (below TCP/UDP)
  - De-facto standard for host level security
  - Developed by the IETF (over many years)
  - Available in most operating systems/devices
    - E.g., XP, Vista, OS X, Linux, BSD*, …
  - Implements a wide range of protocols and cryptographic algorithms
- *Selectively* provides ….
  - Confidentiality, integrity, authenticity, replay protection, DOS protection

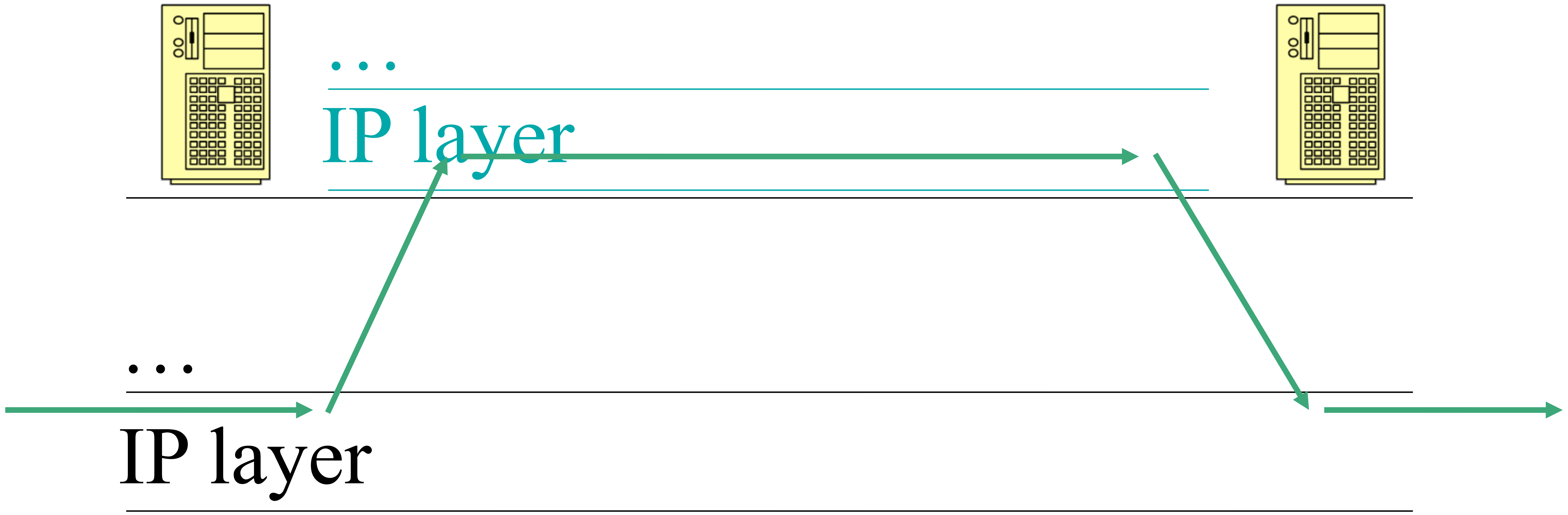
IPsec and the IP protocol stack

- IPsec puts the two main protocols in between IP and the other protocols
  - AH - authentication header
  - ESP - encapsulating security payload
- Other functions provided by external protocols and architectures



Tunneling

- “IP over IP”
  - Network-level packets are encapsulated
  - Allows traffic to evade firewalls

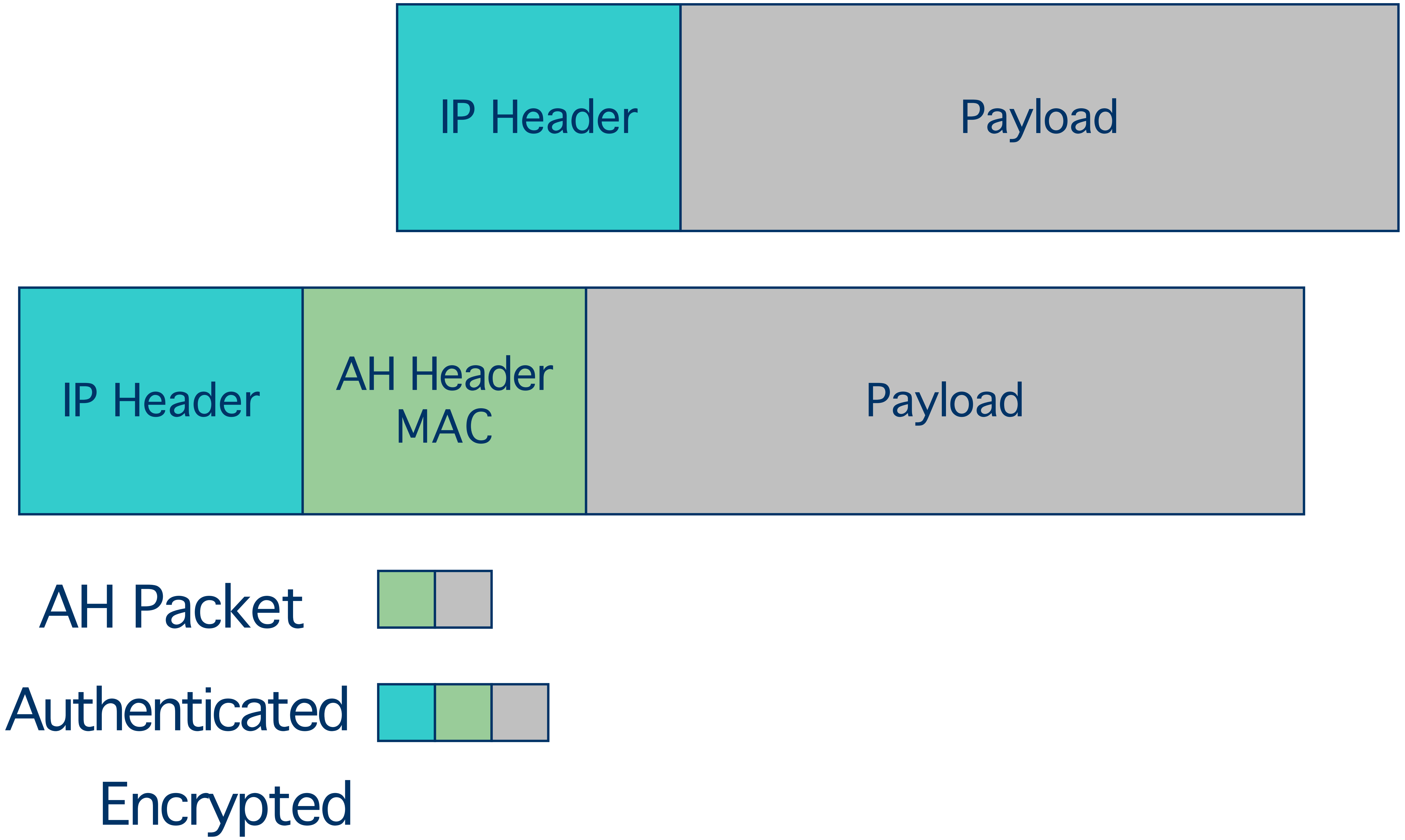


Authentication Header (AH)

- Authenticity and integrity:
  - Provides data integrity by using an authentication algorithm.
    - via HMAC
    - over IP headers and data
  - It does not encrypt the packet.
- Advantage: the authenticity of data and IP header information is protected
  - it gets a little complicated with *mutable* fields (e.g., TTL), which are supposed to be altered by network as packet traverses the network
  - some fields are *immutable*, and are protected
- Confidentiality of data is *not* preserved
- Replay protection via AH sequence numbers
  - note that this replicates some features of TCP (good?)

Authentication Header (AH)

- Modifications to the packet format

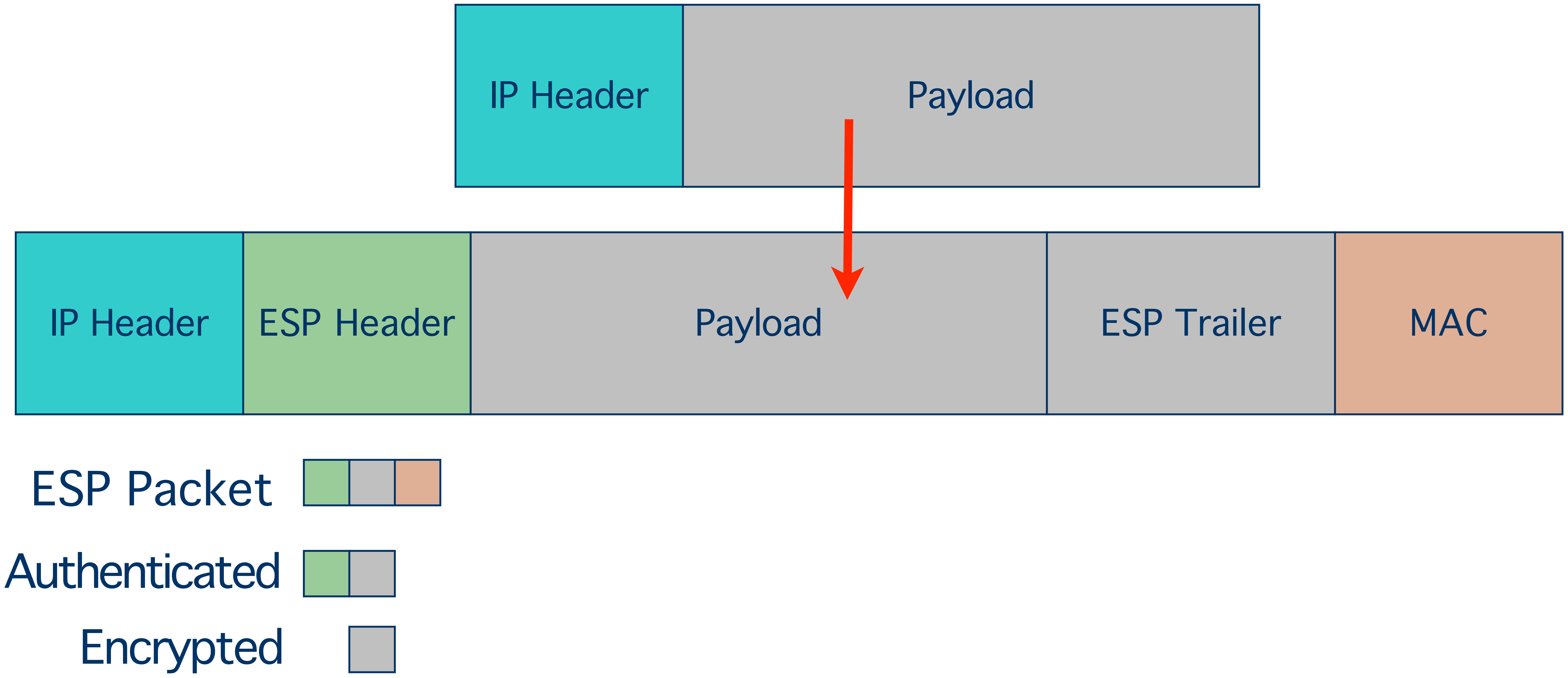


Encapsulating Security Payload (ESP)

- Confidentiality, authenticity and integrity
  - via encryption and HMAC
  - over IP *payload* (data)
- Advantage: the security manipulations are done solely on user data
  - TCP packet is fully secured
  - simplifies processing
- Use "null" encryption to get authenticity/integrity only
- Note that the TCP ports are hidden when encrypted
  - good: better security, less is known about traffic
  - bad: impossible for FW to filter/traffic based on port
- Cost: can require many more resources than AH

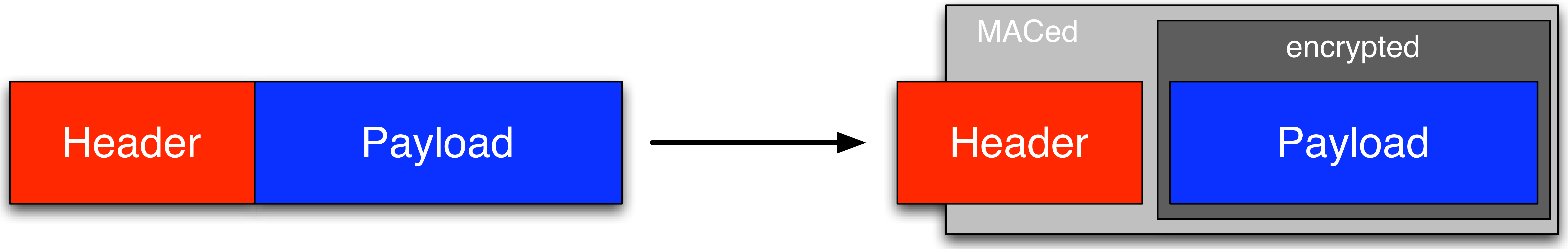
Encapsulating Security Payload (ESP)

- Modifications to packet format

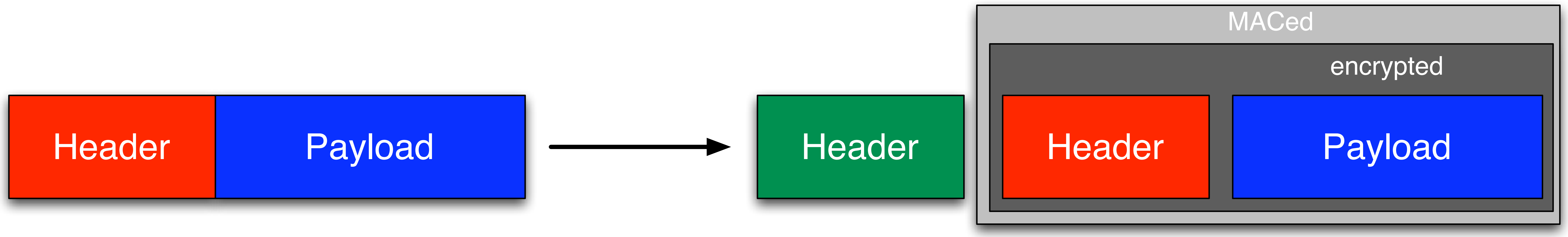


Modes of operation

- Transport : the payload is encrypted and the non-mutable fields are integrity verified (via MAC)

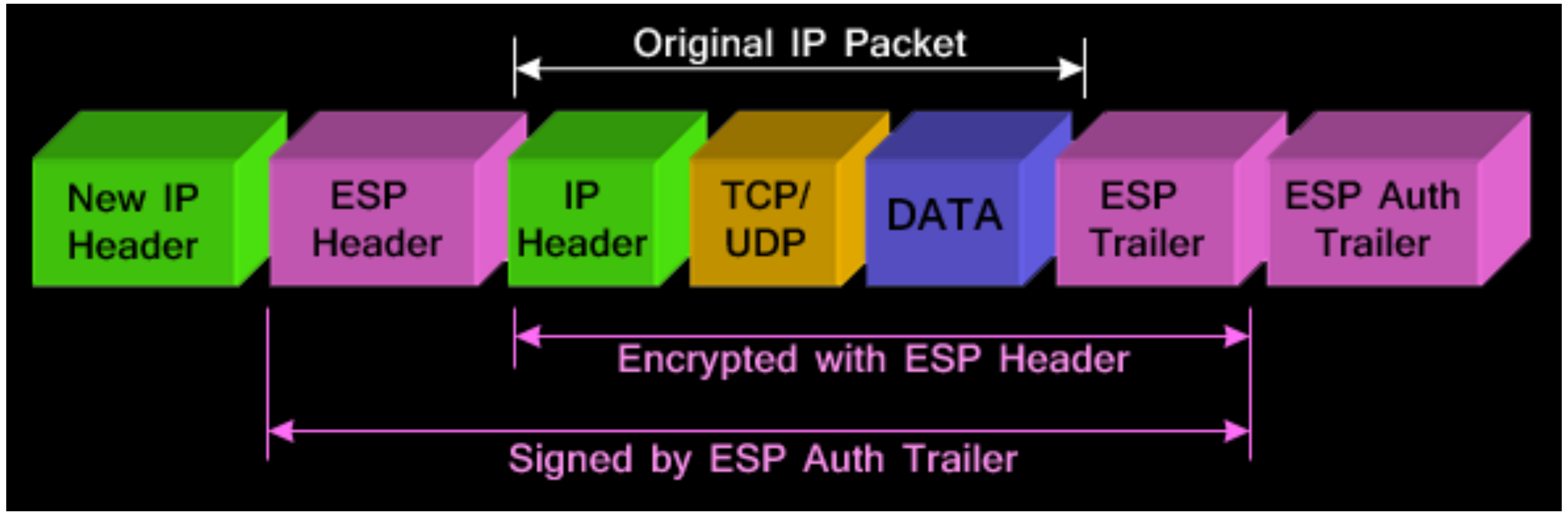


- Tunnel : each packet is completely encapsulated (encrypted) in an outer IP packet
  - Hides not only data, but some routing information

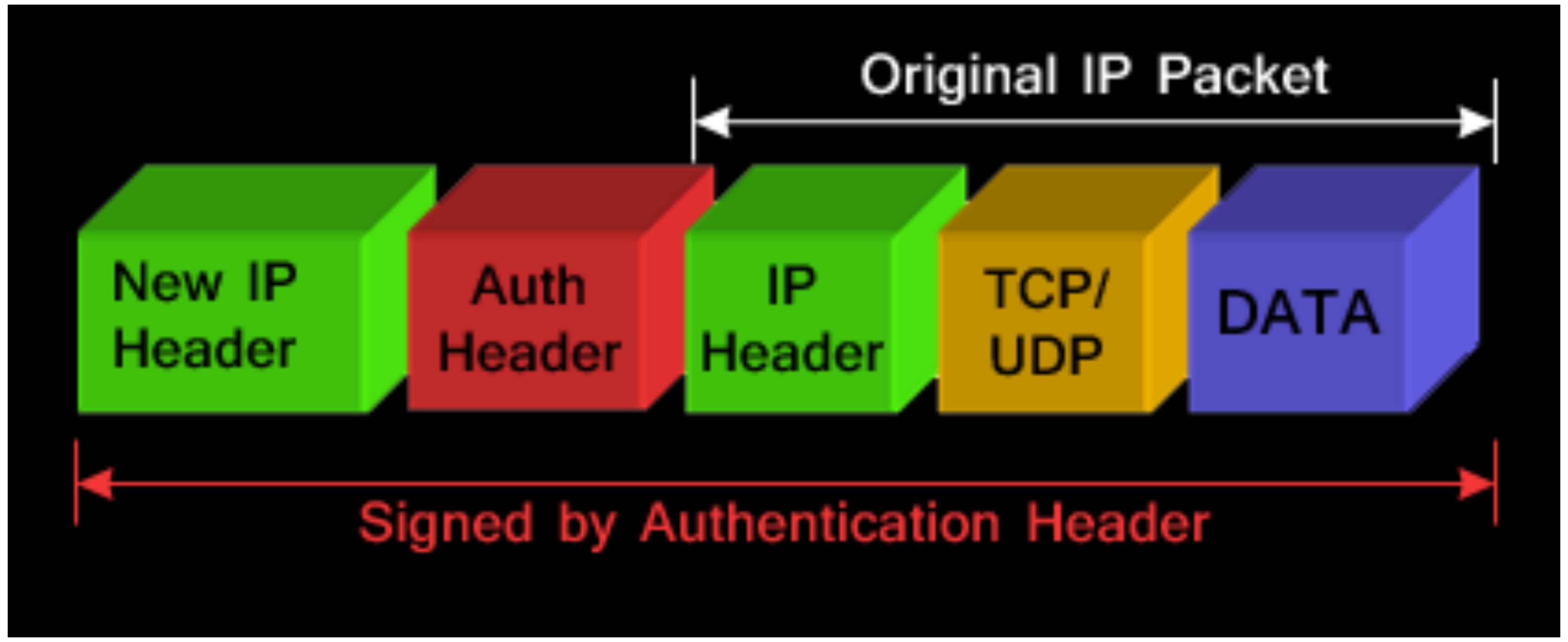


Tunnel mode with ESP and AH

IPSec Tunnel mode with ESP header:

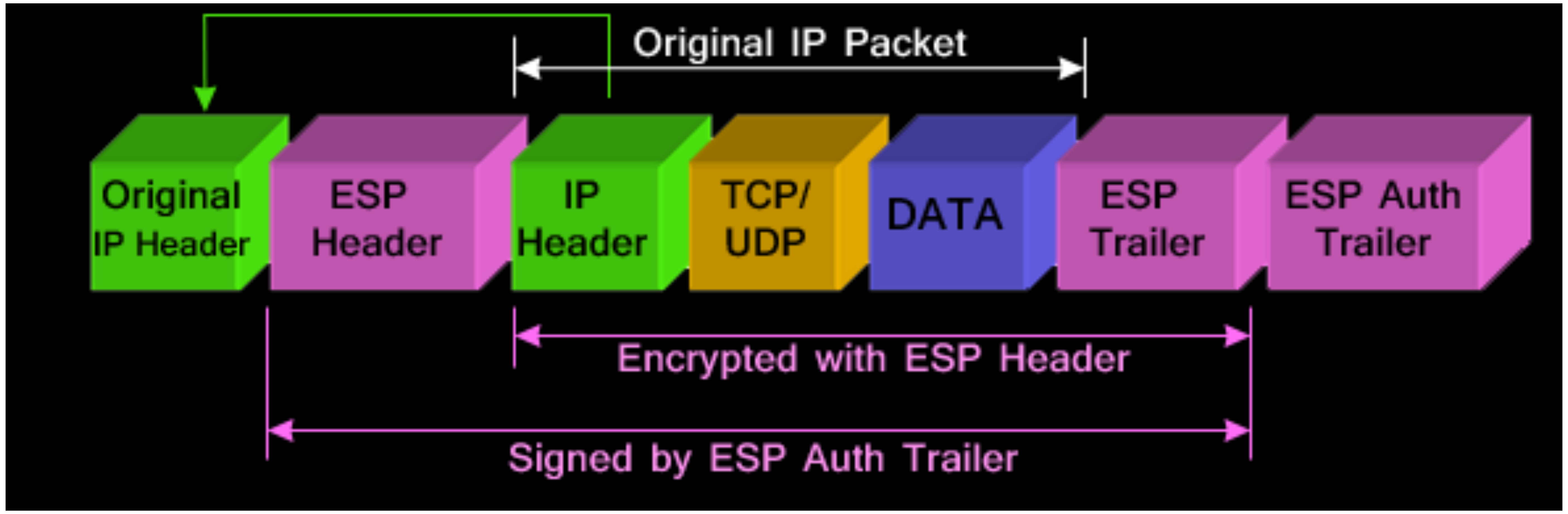


IPSec Tunnel mode with AH header:

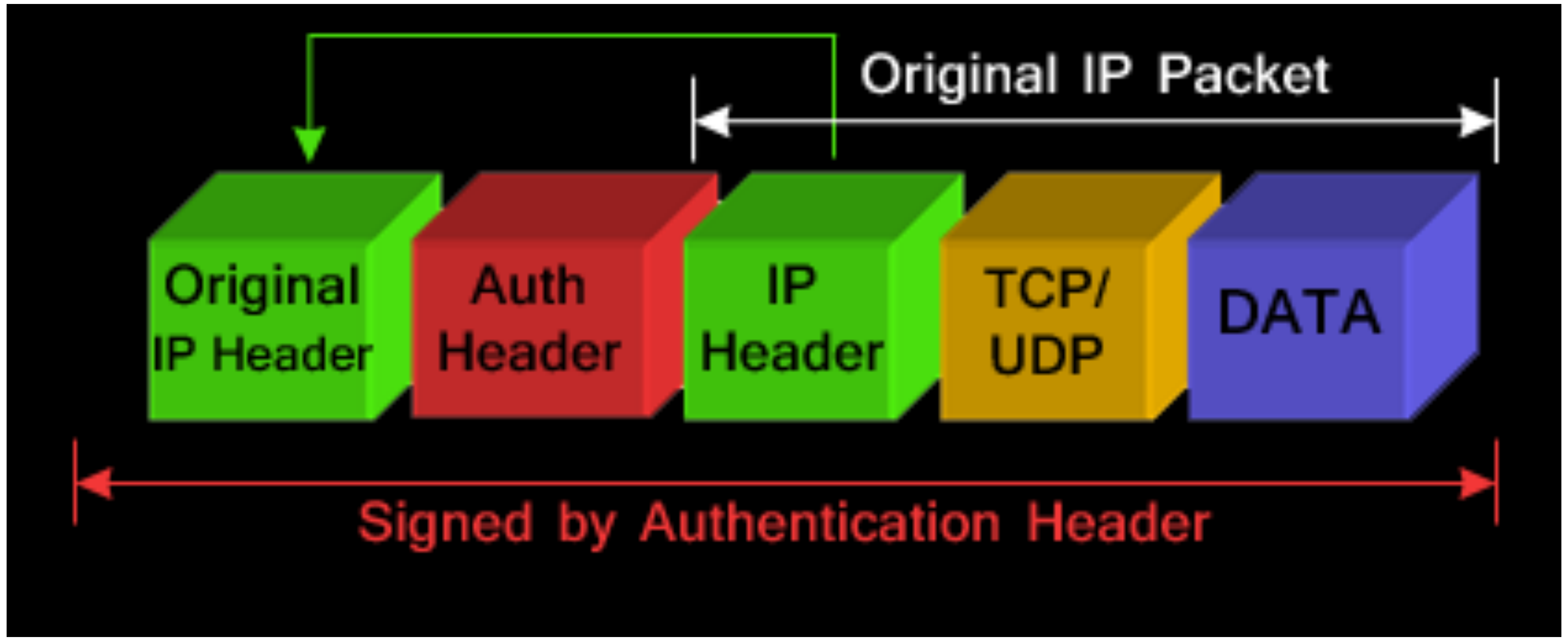


Transport mode with ESP and AH

IPSec Transport mode with ESP header:



IPSec Tunnel mode with AH header:



Tunnel and Transport Mode

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

Practical Issues and Limitations

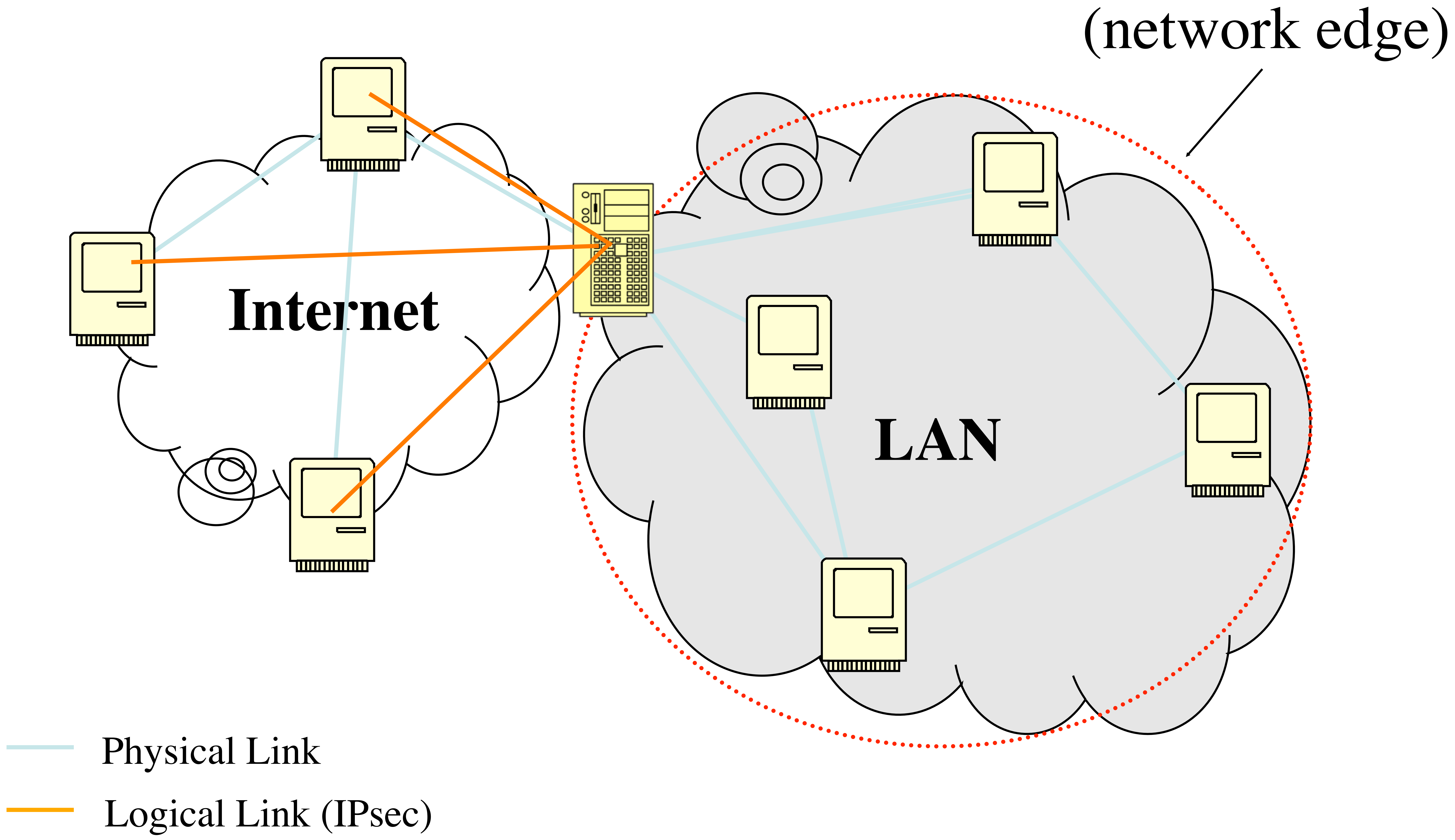
- IPsec implementations
  - Large footprint
    - resource poor devices are in trouble
    - New standards to simplify (e.g, JFK, IKE2)
  - Slow to adopt new technologies
  - Configuration is really complicated/obscure

- Issues
  - IPsec tries to be "everything for everybody at all times"
    - Massive, complicated, and unwieldy
  - Policy infrastructure has not emerged
  - Large-scale management tools are limited (e.g., CISCO)
  - Often not used securely (common pre-shared keys)

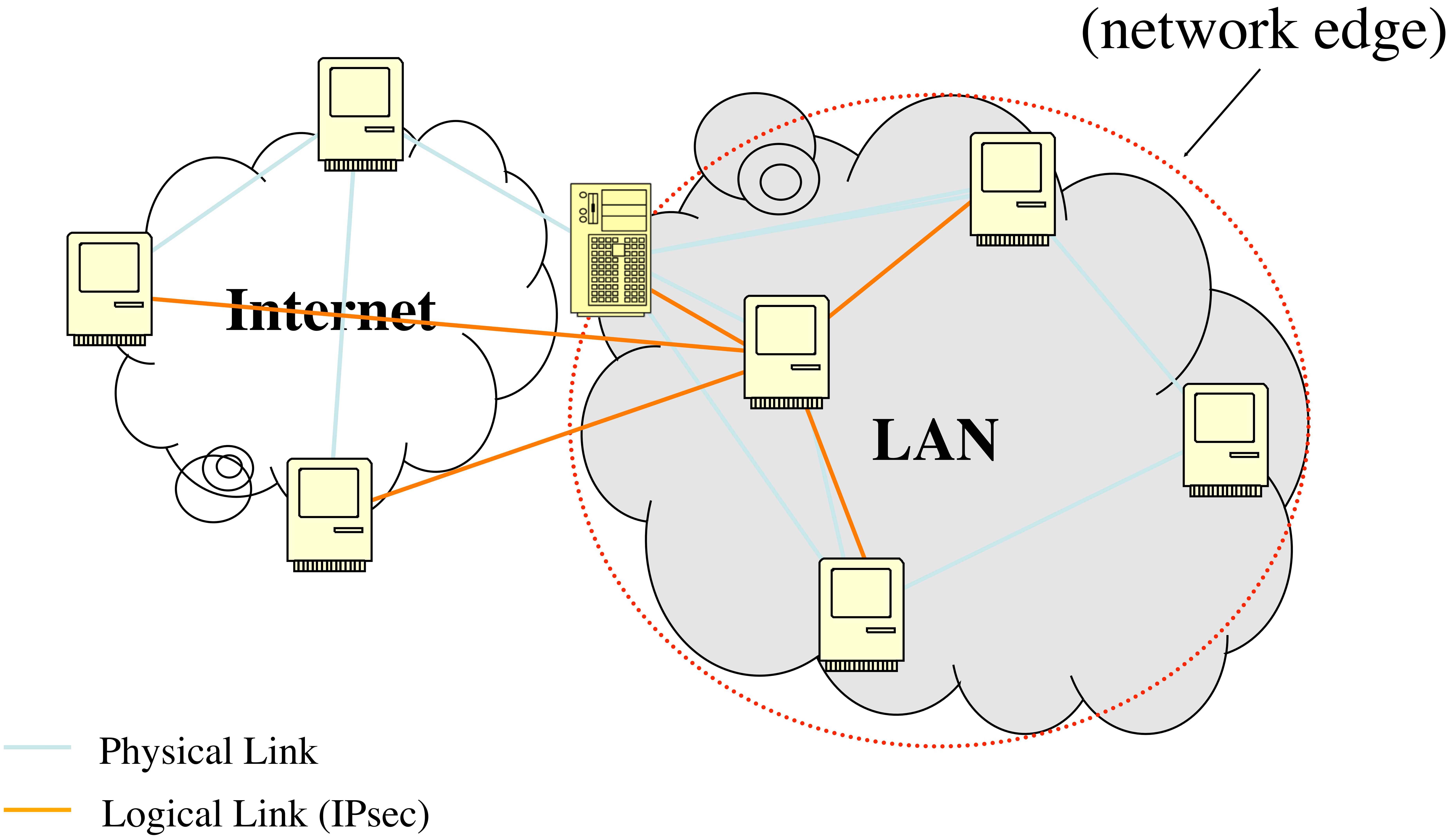
Network Isolation: VPNs

- Idea: I want to create a collection of hosts that operate in a coordinated way
  - E.g., a virtual security perimeter over physical network
  - Hosts work as if they are isolated from malicious hosts
- Solution: Virtual Private Networks
  - Create virtual network topology over physical network
  - Use communications security protocol suites to secure virtual links "tunneling"
  - Manage networks as if they are physically separate
  - Hosts can route traffic to regular networks (*split-tunneling*)

VPN Example: RW/Telecommuter



VPN Example: Hub and Spoke



VPN Example: Mesh