Network Security

Asim Rasheed

Where we are …

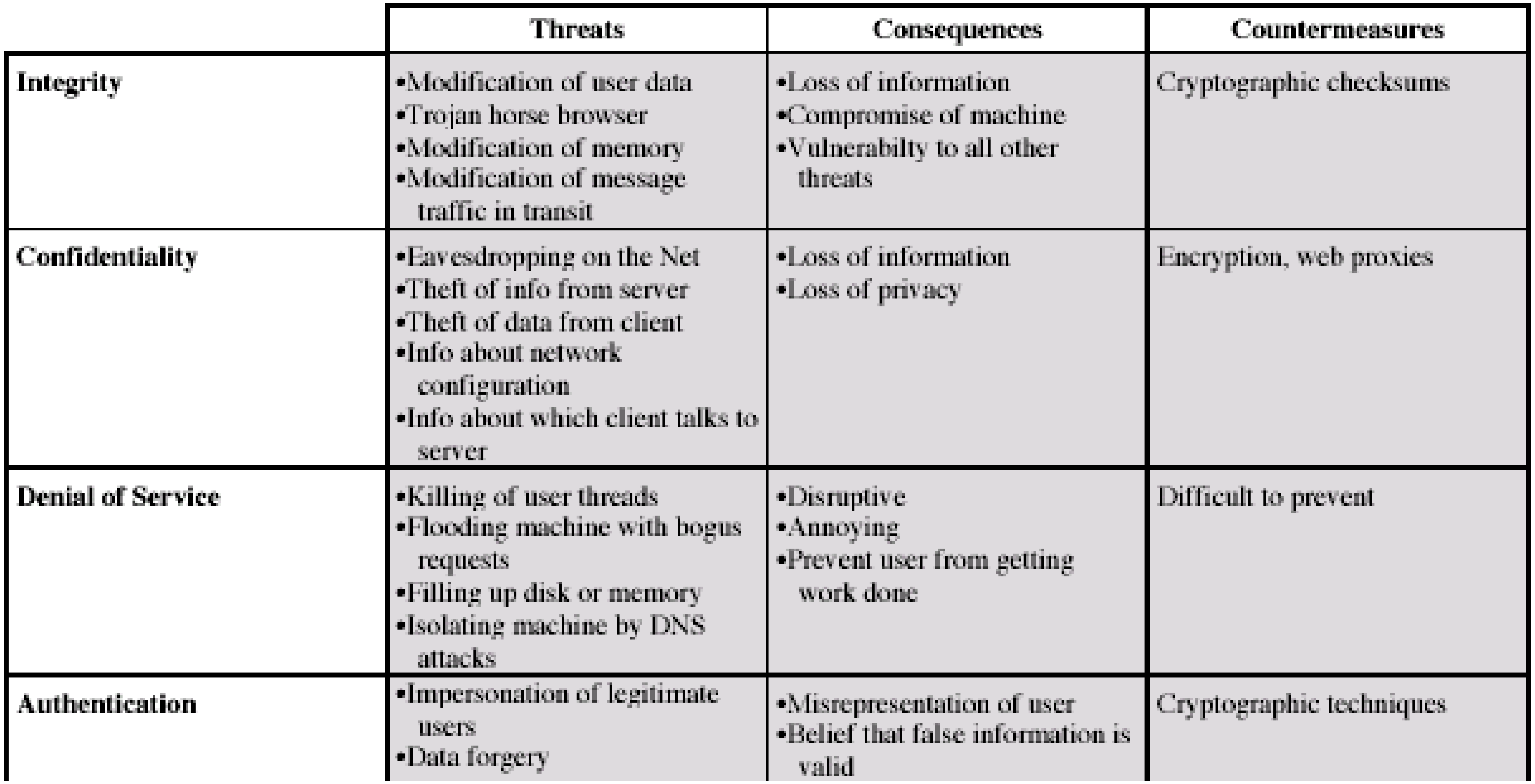
- Introduction to network security
- Vulnerabilities in IP
- I. CRYPTOGRAPHY
  - Symmetric Encryption and Message Confidentiality
  - Public-Key Cryptography and Message Authentication
- II. NETWORK SECURITY APPLICATIONS
  - Authentication Applications (Kerberos, X.509)
  - Electronic Mail Security (PGP, S/MIME)
  - IP Security (IPSec, AH, ESP, IKE)
  - Web Security (SSL, TLS, SET)
- III. SYSTEM SECURITY
  - Intruders and intrusion detection
  - Malicious Software (viruses)
  - Firewalls and trusted systems

Secure Socket Layer (SSL)

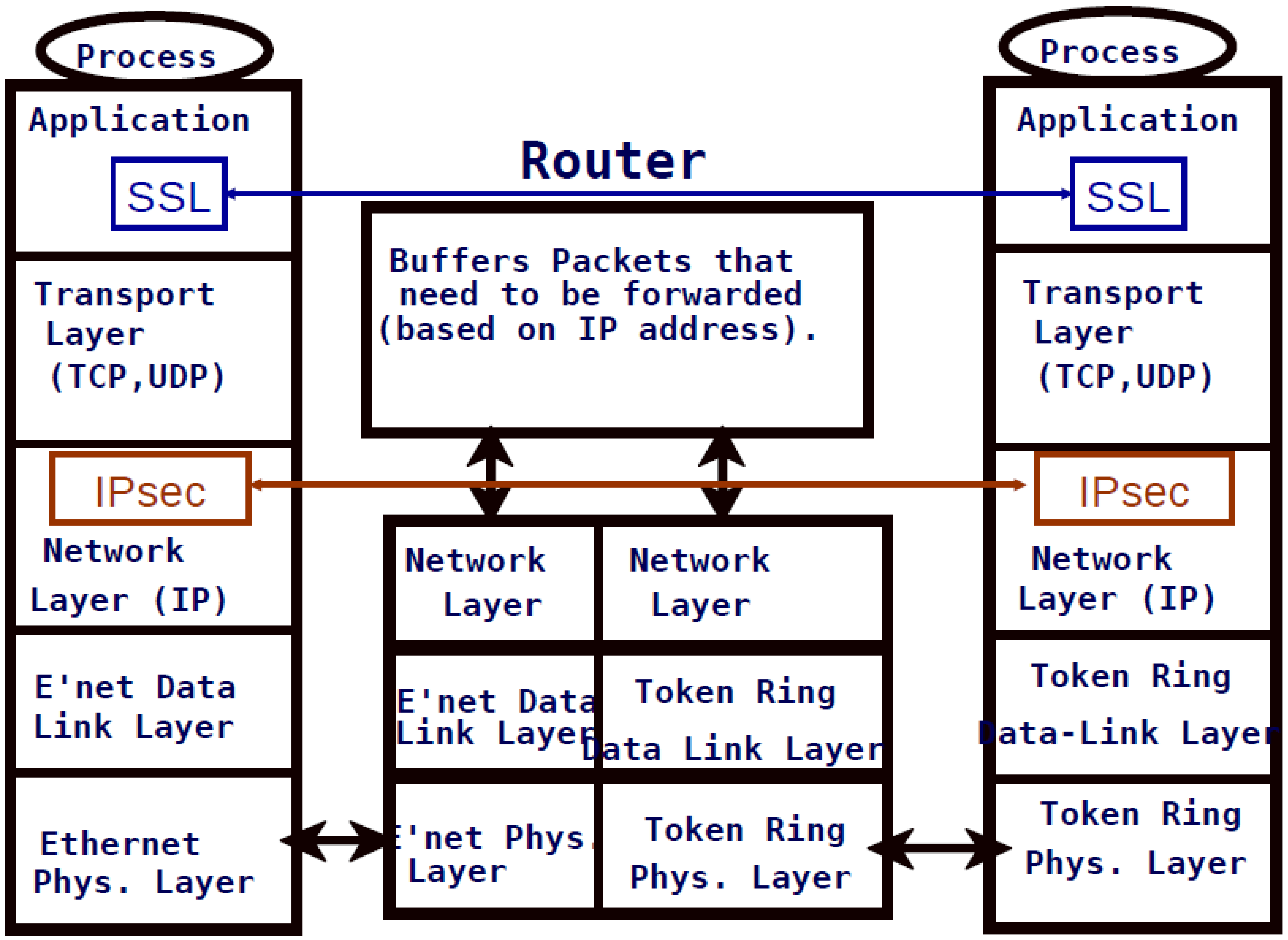
Web Security

- Web now widely used by business, government, individuals
- But Internet & Web are vulnerable
- Have a variety of threats
  - integrity
  - confidentiality
  - denial of service
  - authentication
- Need added security mechanisms

Comparison of Threats on Web



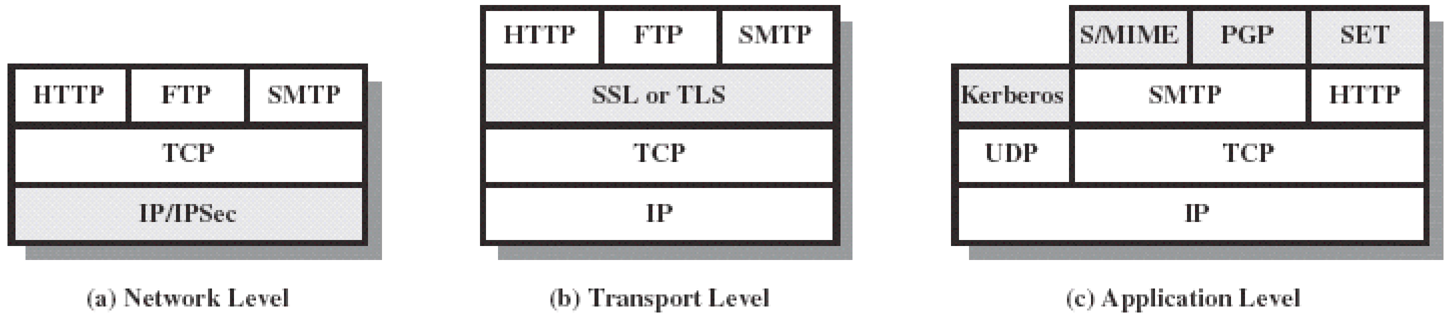
	Threats	Consequences	Countermeasures
Integrity	•Modification of user data •Trojan horse browser •Modification of memory •Modification of message traffic in transit	•Loss of information •Compromise of machine •Vulnerabilty to all other threats	Cryptographic checksums
Confidentiality	•Eavesdropping on the Net •Theft of info from server •Theft of data from client •Info about network configuration •Info about which client talks to server	•Loss of information •Loss of privacy	Encryption, web proxies
Denial of Service	•Killing of user threads •Flooding machine with bogus requests •Filling up disk or memory •Isolating machine by DNS attacks	•Disruptive •Annoying •Prevent user from getting work done	Difficult to prevent
Authentication	•Impersonation of legitimate users •Data forgery	•Misrepresentation of user •Belief that false information is valid	Cryptographic techniques



Process
Application
SSL
Transport Layer (TCP,UDP)
IPsec
Network Layer (IP)
E'net Data Link Layer
Ethernet Phys. Layer
Router
Buffers Packets that need to be forwarded (based on IP address).
Network Layer
Network Layer
E'net Data Link Layer
Token Ring Data Link Layer
E'net Phys. Layer
Token Ring Phys. Layer
Process
Application
SSL
Transport Layer (TCP,UDP)
IPsec
Network Layer (IP)
Token Ring Data-Link Layer
Token Ring Phys. Layer

Relative security facilities in TCP/IP

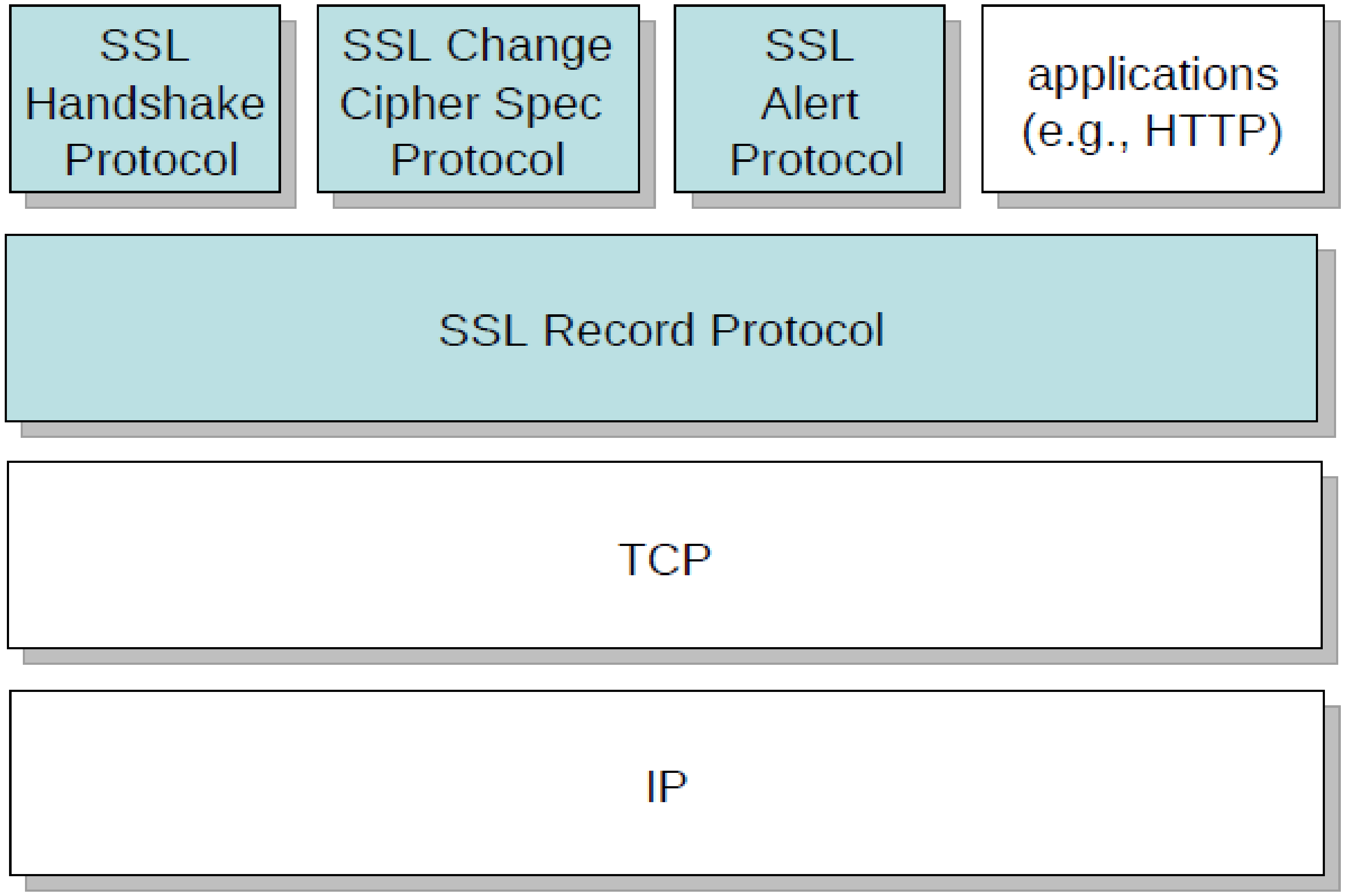
- SSL/TLS is Transport Layer Security (is not "IPsec Transport Level Security")?
- SSL/TLS is used for email (SMTP/TLS or POP/TLS or IMAP/TLS)
- SSL/TLS is used for secure Web access (HTTPS)
- Secure Shell, SSH, is Telnet + SSL + other features
- Secure Copy, SCP, copies files using SSH (no other FTP functions)



What are SSL and TLS?

- SSL – Secure Socket Layer
- TLS – Transport Layer Security
- Both provide a secure transport connection between applications (e.g, web server & browser)
- SSL was developed by Netscape
- SSL version 3.0 has been implemented in many web browsers (e.g., Netscape Navigator and IE) and web servers and widely used on the Internet
- SSL v3.0 was specified in an Internet Draft ,1996
  - Evolved into TLS specified in RFC 2246
  - TLS can be viewed as SSL v3.1

SSL architecture



MAC

- Similar to HMAC but the pads are concatenated
- Supported hash functions:
  - MD5
  - SHA-1
- pad_1 is 0x36 repeated 48 times (MD5) or 40 times (SHA-1)
- pad_2 is 0x5C repeated 48 times (MD5) or 40 times (SHA-1)

```
MAC = hash( MAC_write_secret | pad_2 |
                hash( MAC_write_secret | pad_1 |
   seq_num | type | length | fragment ) )
```

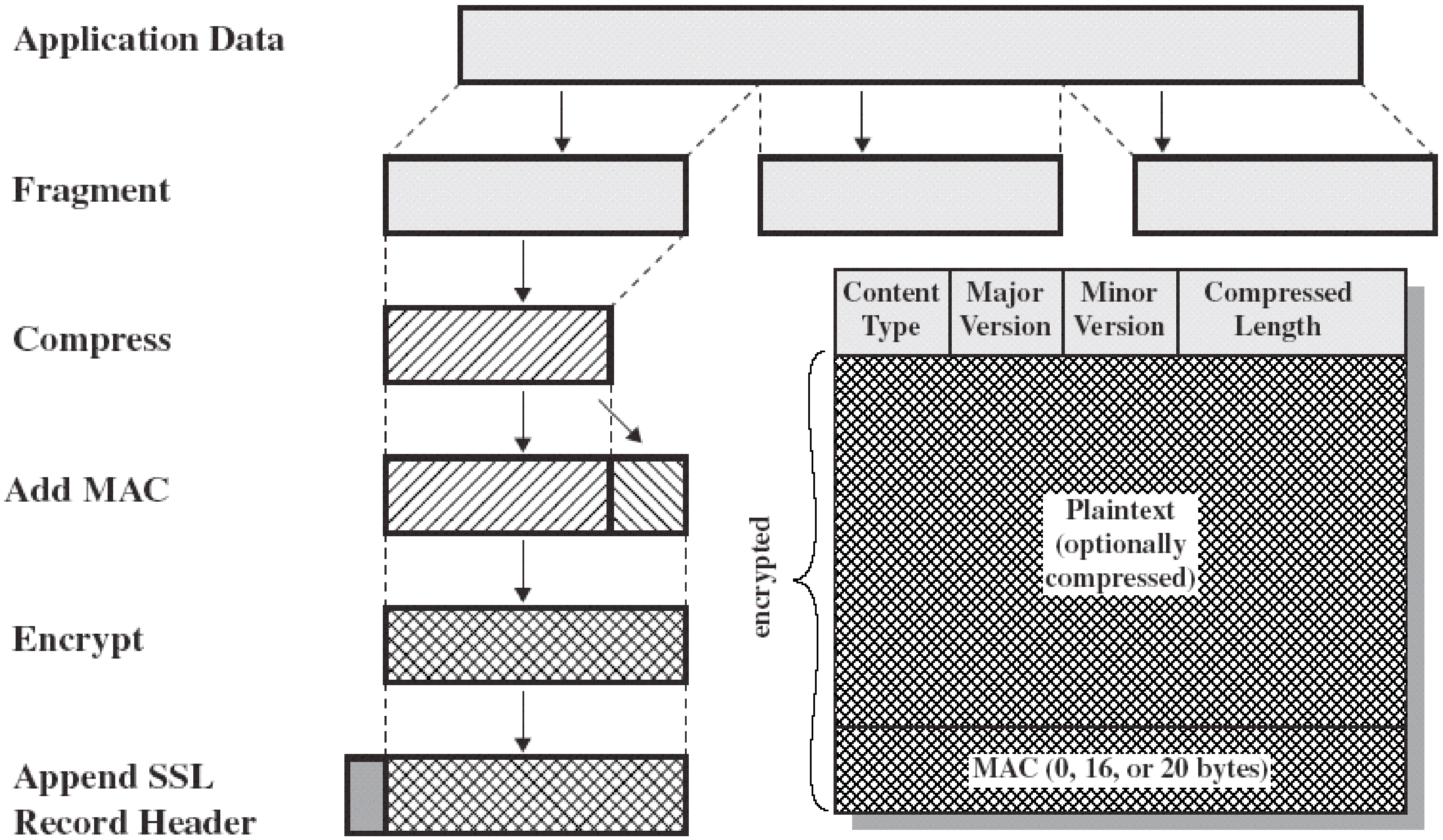
Encryption- Supported Algorithms

- Block ciphers (in CBC mode)
  - RC2_40
  - DES_40 & DES_56
  - 3DES_168
  - IDEA_128
  - FORTEZZA_80
- Stream ciphers
  - RC4_40
  - RC4_128
- If a block cipher is used, than padding is applied
  - last byte of the padding is the padding length

SSL components

- SSL Handshake Protocol
  - negotiation of security algorithms and parameters
  - key exchange
  - server authentication and optionally client authentication
- SSL Record Protocol
  - fragmentation
  - compression
  - message authentication and integrity protection
  - encryption

SSL Record Protocol Operation



SSL components

- SSL Alert Protocol
  - error messages (fatal alerts and warnings)
- SSL Change Cipher Spec Protocol
  - a single message that indicates the end of the SSL handshake

SSL Sessions

- An association between a client and a server
- Sessions are stateful; the session state includes security algorithms and parameters
- Session may include multiple secure connections between the same client and server

SSL connections

- Connections of the same session share the session state
- Sessions are used to avoid expensive negotiation of new security parameters for each connection
- There may be multiple simultaneous sessions between the same two parties, but this feature is not used in practice

Session and Connection States

- Session identifier
  - arbitrary byte sequence chosen by the server to identify the session
- Peer certificate
  - X.509 certificate of the peer
  - may be null
- Compression method
- Cipher spec
  - bulk data encryption algorithm (e.g., null, DES, 3DES, …)
  - MAC algorithm (e.g., MD5, SHA-1)
  - cryptographic attributes (e.g., hash size, IV size, …)

Session States….

- Master secret
  - 48-byte secret shared between the client and the server
- Is resume able
  - a flag indicating whether the session can be used to initiate new connections

Connection States

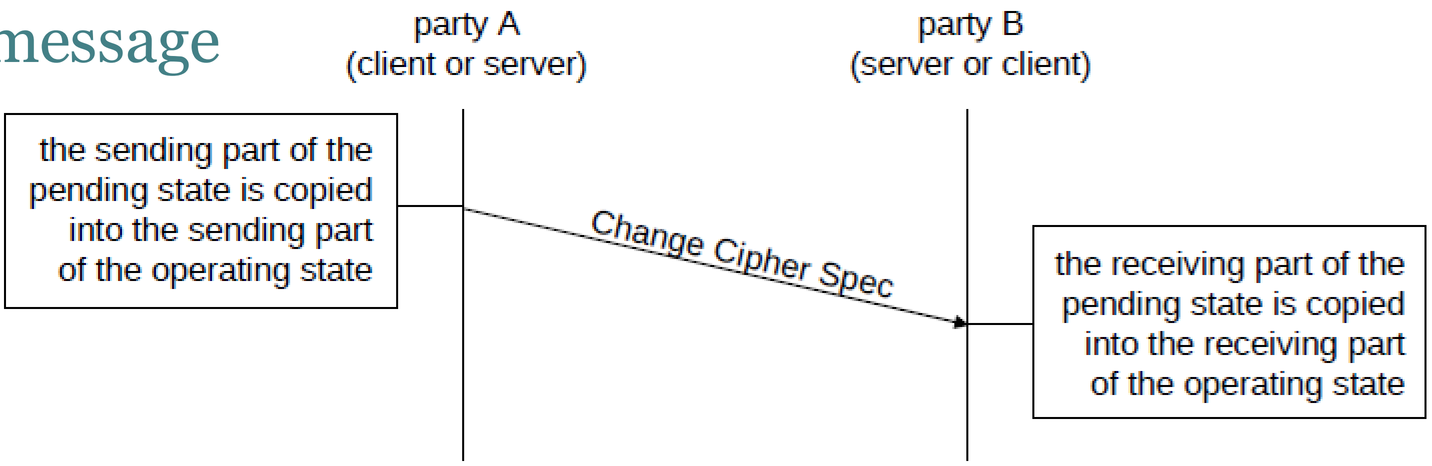
- Server and client random
  - random byte sequences chosen by the server and the client for every connection
- Server write MAC secret
  - secret key used in MAC operations on data sent by the server
- Client write MAC secret
  - secret key used in MAC operations on data sent by the client
- Server write key
  - secret encryption key for data encrypted by server

Connection States

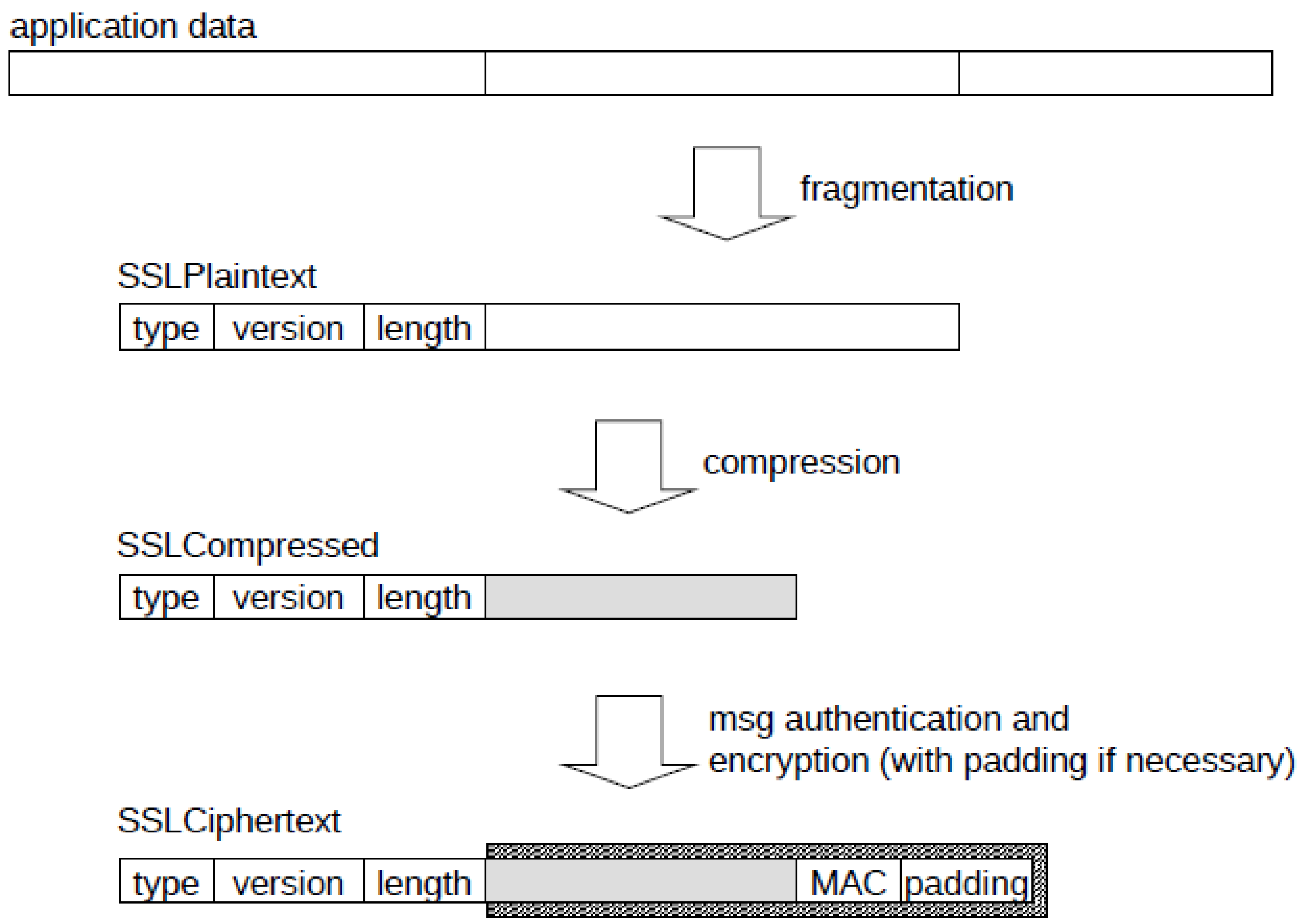
- Client write key
  - secret encryption key for data encrypted by client
- Initialization vectors
  - an IV is maintained for each encryption key if CBC mode is used
  - initialized by the SSL Handshake Protocol
  - final ciphertext block from each record is used as IV with the following record
- Sending and receiving sequence numbers
  - sequence numbers are 64 bits long
  - reset to zero after each Change Cipher Spec message

State changes

- Operating state
  - currently used state
- Pending state
  - state to be used
  - built using the current state
- Operating state =>Pending state
  - at the transmission and reception of a Change Cipher Spec message


party A
(client or server)
party B
(server or client)
the sending part of the pending state is copied into the sending part of the operating state
Change Cipher Spec
the receiving part of the pending state is copied into the receiving part of the operating state

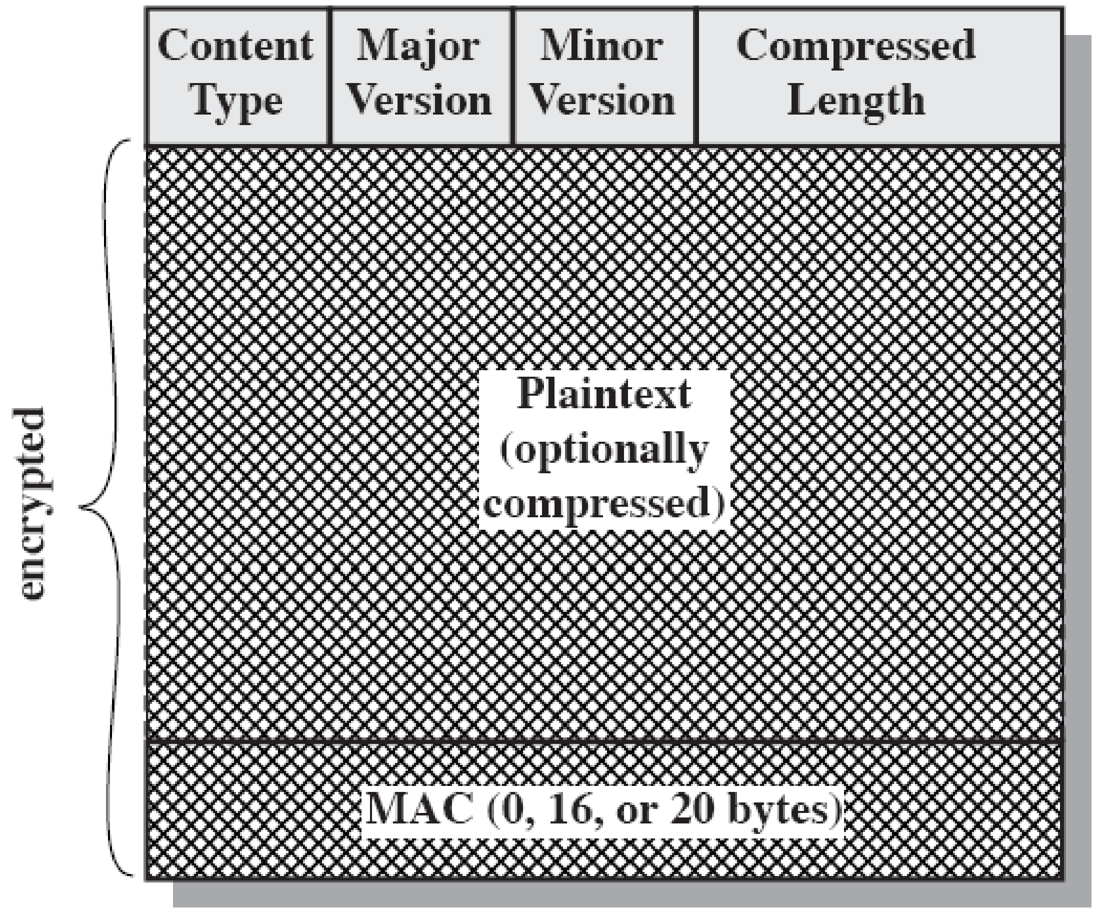
SSL Record Protocol – processing overview



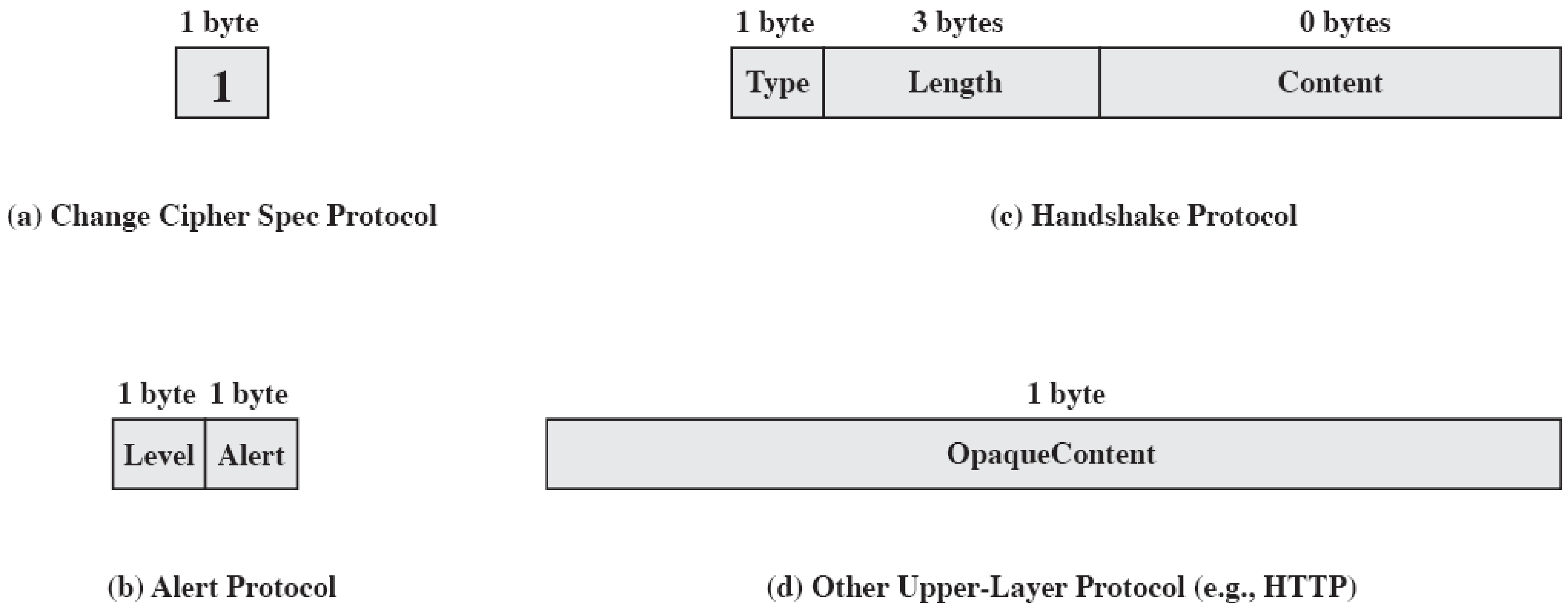
Header

- type
  - the higher level protocol used to process the enclosed fragment
  - possible types:
    - change_cipher_spec
    - alert
    - handshake
    - application_data
- SSL version, currently 3.0
- length
  - length (in bytes) of enclosed or compressed fragment
  - max value is $2^{14} + 2048$

SSL Record Format



SSL Record Protocol Payload



Record Layer

- The SSL Record Layer receives uninterrupted data from higher layers in non-empty blocks of arbitrary size

Fragmentation – Record Layer

- The record layer fragments information blocks into SSLPlaintext records of 2^14 bytes or less.
- Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType may be coalesced into a single SSLPlaintext record).

Record compression and decompression – Record Layer

- All records are compressed using the compression algorithm defined in the current session state
- There is always an active compression algorithm, however initially it is defined as CompressionMethod.null.
- The compression algorithm translates an SSLPlaintext structure into an SSLCompressed structure

Record compression and decompression – Record Layer…..

- Compression functions erase their state information whenever the CipherSpec is replaced.
- Note
  - The CipherSpec is part of the session state
- Compression must be lossless and may not increase the content length by more than 1024 bytes.
- If the decompression function encounters an SSLCompressed.fragment that would decompress to a length in excess of 2^14 bytes, it should issue a fatal decompression_failure alert

Record payload protection and the CipherSpec

- All records are protected using the encryption and MAC algorithms defined in the current CipherSpec.
- There is always an active CipherSpec, however initially it is SSL_NULL_WITH_NULL_NULL, which does not provide any security.
- Once the handshake is complete, the two parties have shared secrets which are used to encrypt records and compute keyed message authentication codes (MACs) on their contents

Record payload protection and the CipherSpec…..

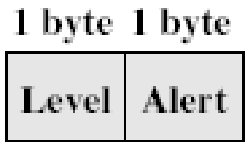
- Techniques used to perform the encryption and MAC operations are defined by the CipherSpec
- The encryption and MAC functions translate an SSLCompressed structure into a SSLCiphertext.
- The decryption functions reverse the process.
- Transmissions also include a sequence number so that missing, altered, or extra messages are detectable.
- Note that the MAC is computed before encryption. The stream cipher encrypts the entire block, including the MAC

SSL Alert Protocol

- One of the content types supported by the SSL Record layer is the alert type.
- Alert messages convey the severity of the message and a description of the alert.
- Alert messages with a level of fatal result in the immediate termination of the connection. In this case, other connections corresponding to the session may continue, but the session identifier must be invalidated, preventing the failed session from being used to establish new connections.

SSL Alert Protocol

- Like other messages, alert messages are encrypted and compressed, as specified by the current connection state
- Each alert message consists of 2 fields (bytes)
- First field (byte): "warning" or "fatal"
- Second field (byte): "fatal"
  - unexpected_message
  - bad_record_MAC
  - decompression_failure
  - handshake_failure
  - illegal_parameter



SSL Alert Protocol

- warning
  - close_notify
  - no_certificate
  - bad_certificate
  - unsupported_certificate
  - certificate_revoked
  - certificate_expired
  - certificate_unknown
- In case of a fatal alert
  - connection is terminated
  - session ID is invalidated => no new connection can be established within this session

Closure alerts

- The client and the server must share knowledge that the connection is ending in order to avoid a truncation attack. Either party may initiate the exchange of closing messages.
- Either party may initiate a close by sending a close_notify alert. Any data received after a closure alert is ignored.

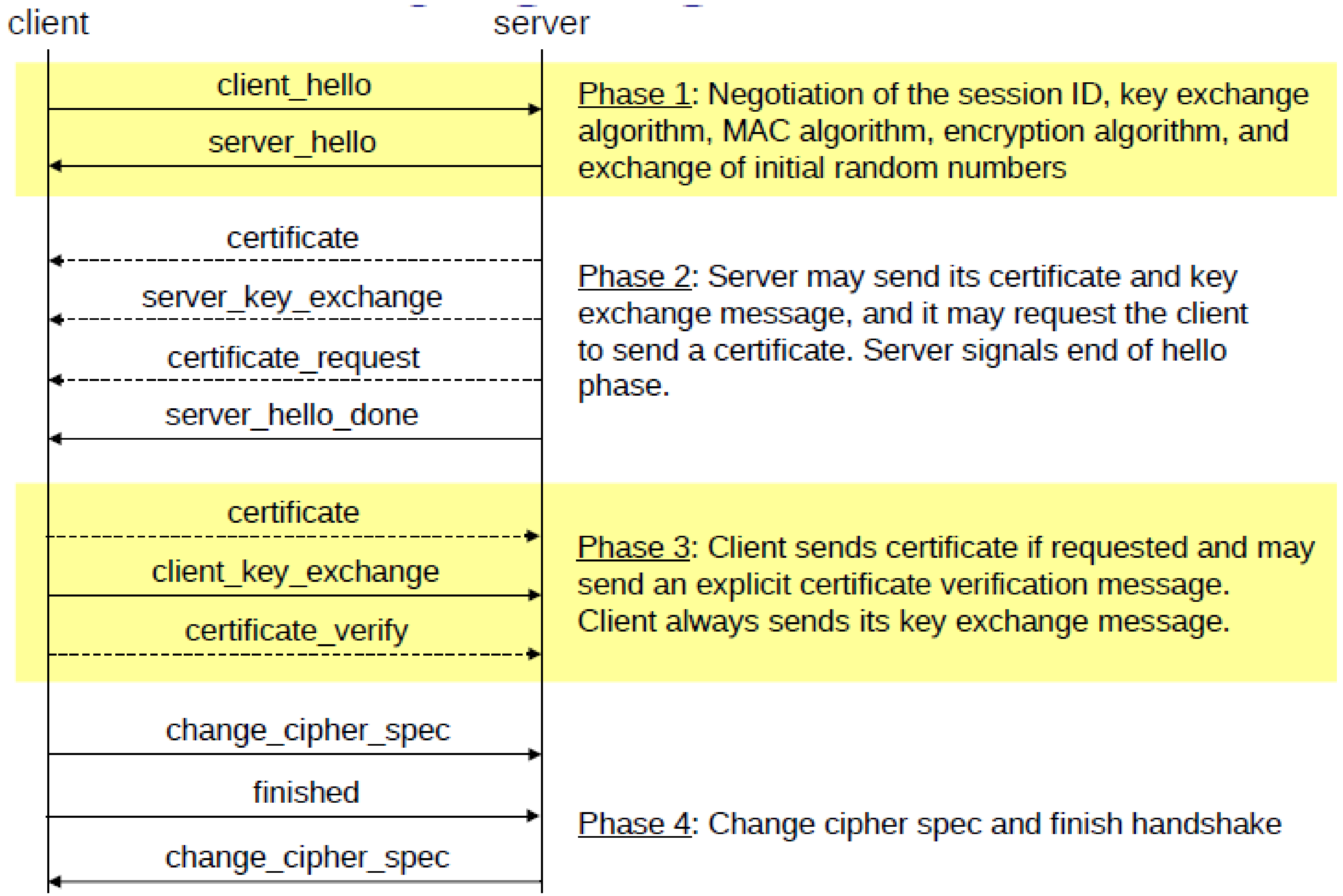
Closure alerts…

- Each party is required to send a close_notify alert before closing the write side of the connection. It is required that the other party respond with a close_notify alert of its own and close down the connection immediately, discarding any pending writes
- It is not required for the initiator of the close to wait for the responding close_notify alert before closing the read side of the connection.
- It is assumed that closing a connection reliably delivers pending data before destroying transport

Error alerts

- Error handling in the SSL Handshake protocol is very simple.
- When an error is detected, the detecting party sends a message to the other party.
- Upon transmission or receipt of an fatal alert message, both parties immediately close the connection.
- Servers and clients are required to forget any session-identifiers, keys, and secrets associated with a failed connection

SSL Handshake Protocol – overview



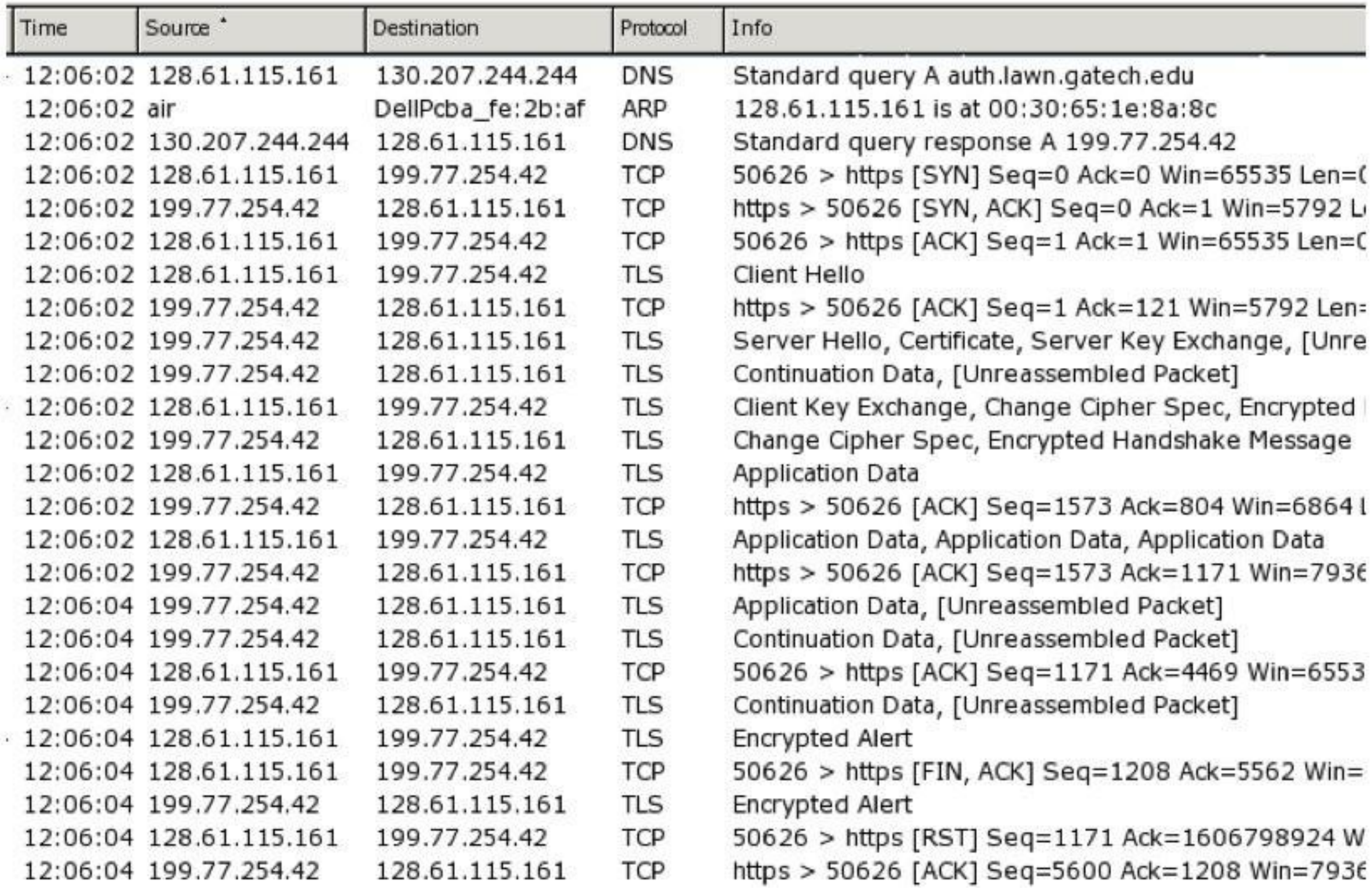
Phase 1: Negotiation of the session ID, key exchange algorithm, MAC algorithm, encryption algorithm, and exchange of initial random numbers

Phase 2: Server may send its certificate and key exchange message, and it may request the client to send a certificate. Server signals end of hello phase.

Phase 3: Client sends certificate if requested and may send an explicit certificate verification message. Client always sends its key exchange message.

Phase 4: Change cipher spec and finish handshake

WireShark* View of HTTPS (TLS = SSL) Connection



Time	Source	Destination	Protocol	Info
12:06:02	128.61.115.161	130.207.244.244	DNS	Standard query A auth.lawn.gatech.edu
12:06:02	air	DellPcba_fe:2b:af	ARP	128.61.115.161 is at 00:30:65:1e:8a:8c
12:06:02	130.207.244.244	128.61.115.161	DNS	Standard query response A 199.77.254.42
12:06:02	128.61.115.161	199.77.254.42	TCP	50626 > https [SYN] Seq=0 Ack=0 Win=65535 Len=
12:06:02	199.77.254.42	128.61.115.161	TCP	https > 50626 [SYN, ACK] Seq=0 Ack=1 Win=5792 L
12:06:02	128.61.115.161	199.77.254.42	TCP	50626 > https [ACK] Seq=1 Ack=1 Win=65535 Len=
12:06:02	128.61.115.161	199.77.254.42	TLS	Client Hello
12:06:02	199.77.254.42	128.61.115.161	TCP	https > 50626 [ACK] Seq=1 Ack=121 Win=5792 Len=
12:06:02	199.77.254.42	128.61.115.161	TLS	Server Hello, Certificate, Server Key Exchange, [Unre
12:06:02	199.77.254.42	128.61.115.161	TLS	Continuation Data, [Unreassembled Packet]
12:06:02	128.61.115.161	199.77.254.42	TLS	Client Key Exchange, Change Cipher Spec, Encrypted
12:06:02	199.77.254.42	128.61.115.161	TLS	Change Cipher Spec, Encrypted Handshake Message
12:06:02	128.61.115.161	199.77.254.42	TLS	Application Data
12:06:02	199.77.254.42	128.61.115.161	TCP	https > 50626 [ACK] Seq=1573 Ack=804 Win=6864
12:06:02	128.61.115.161	199.77.254.42	TLS	Application Data, Application Data, Application Data
12:06:02	199.77.254.42	128.61.115.161	TCP	https > 50626 [ACK] Seq=1573 Ack=1171 Win=793
12:06:04	199.77.254.42	128.61.115.161	TLS	Application Data, [Unreassembled Packet]
12:06:04	199.77.254.42	128.61.115.161	TLS	Continuation Data, [Unreassembled Packet]
12:06:04	128.61.115.161	199.77.254.42	TCP	50626 > https [ACK] Seq=1171 Ack=4469 Win=6553
12:06:04	199.77.254.42	128.61.115.161	TLS	Continuation Data, [Unreassembled Packet]
12:06:04	128.61.115.161	199.77.254.42	TLS	Encrypted Alert
12:06:04	128.61.115.161	199.77.254.42	TCP	50626 > https [FIN, ACK] Seq=1208 Ack=5562 Win=
12:06:04	199.77.254.42	128.61.115.161	TLS	Encrypted Alert
12:06:04	128.61.115.161	199.77.254.42	TCP	50626 > https [RST] Seq=1171 Ack=1606798924 W
12:06:04	199.77.254.42	128.61.115.161	TCP	https > 50626 [ACK] Seq=5600 Ack=1208 Win=793

Hello Messages- Client

- client_hello
  - client_version
    - the highest version supported by the client
  - client_random
    - current time (4 bytes) + pseudo random bytes (28 bytes)
  - session_id
    - empty if the client wants to create a new session, or
    - the session ID of an old session within which the client wants to create the new connection

Client Messages

- cipher_suites
  - list of cryptographic options supported by the client ordered by preference
  - a cipher suite contains the specification of the
    - key exchange method, the encryption and the MAC algorithm
    - the algorithms implicitly specify the hash_size, IV_size, and key_material parameters (part of the Cipher Spec of the session state)
  - exmaple: SSL_RSA_with_3DES_EDE_CBC_SHA
- compression_methods
  - list of compression methods supported by the client

Hello Messages - Server

- server_hello
  - server_version
    - min(highest version supported by client, highest version supported by server)
  - server_random
    - current time + random bytes
    - random bytes must be independent of the client random

Server Messages

- session_id chosen by the server
  - if the client wanted to resume an old session:
    - server checks if the session is resumable
    - if so, it responds with the session ID and the parties proceed to the finished messages
  - if the client wanted a new session
    - server generates a new session ID
- cipher_suite
  - single cipher suite selected by the server from the list given by the client
- compression_method
  - single compression method selected by the server

Supported key exchange methods

- RSA based (SSL_RSA_with...)
  - secret key (pre-master secret) is encrypted with the server's public RSA key
  - server's public key is made available to the client during the exchange
- Fixed Diffie-Hellman (SSL_DH_RSA_with... or SSL_DH_DSS_with...)
  - server has fix DH parameters contained in a certificate signed by a CA
  - client may have fix DH parameters certified by a CA or it may send an unauthenticated onetime DH public value in the client_key_exchange message

Supported Key Exchange Methods

- Ephemeral Diffie-Hellman (SSL_DHE_RSA_ with… or SSL_DHE_DSS_with…)
  - both the server and the client generate one-time DH parameters
  - the server signs its DH parameters with its private RSA or DSS key
  - the client may authenticate itself (if requested by the server) by signing the hash of the handshake messages with its private RSA or DSS key

Supported Key Exchange Methods

- Anonymous Diffie-Hellman
  - both the server and the client generate one-time DH parameters
  - they send their parameters to the peer without authentication
- Fortezza
  - Fortezza proprietary key exchange scheme

Server certificate and key exchange messages

- Certificate
  - required for every key exchange method except for anonymous DH
  - contains one or a chain of X.509 certificates (up to a known root CA)
  - may contain
    - public RSA key suitable for encryption, or
    - public RSA or DSS key suitable for signing only, or
    - fix DH parameters

server_key_exchange

- Sent only if the certificate does not contain enough information to complete the key exchange (e.g., Certificate contains an RSA signing key only)
- May contain
  - public RSA key (exponent and modulus), or
  - DH parameters (p, g, public DH value), or
  - Fortezza parameters

server_key_exchange

- Digitally signed
  - if DSS: SHA-1 hash of (client_random | server_random | server_params) is signed
  - if RSA: MD5 hash and SHA-1 hash of (client_random | server_random | server_params) are concatenated and encrypted with the private RSA key

Certificate request and server hello done msgs

- certificate_request
  - sent if the client needs to authenticate itself
  - specifies which type of certificate is requested (rsa_sign, dss_sign, rsa_fixed_dh, dss_fixed_dh, …)

server_hello_done

- Sent to indicate that the server is finished its part of the key exchange
- After sending this message the server waits for client response
- The client should verify that the server provided a valid certificate and the server parameters are acceptable

Client Authentication and Key Exchange

- certificate
  - sent only if requested by the server
  - may contain
    - public RSA or DSS key suitable for signing only, or
    - fix DH parameters
    - client_key_exchange
  - always sent (but it is empty if the key exchange method is fix DH)
  - may contain
    - RSA encrypted pre-master secret, or
    - client one-time public DH value, or
    - Fortezza key exchange parameters

Client Authentication and Key Exchange….

- certificate_verify
  - sent only if the client sent a certificate
  - provides client authentication
  - contains signed hash of all the previous handshake messages
    - if DSS: SHA-1 hash is signed
    - if RSA: MD5 and SHA-1 hash is concatenated and encrypted with the private key
- MD5(master_secret | pad_2 | MD5(handshake_messages | master_secret | pad_1))
- SHA(master_secret | pad_2 | SHA(handshake_messages | master_secret | pad_1))

Finished messages

- finished
  - sent immediately after the change_cipher_spec message
  - first message that uses the newly negotiated algorithms, keys, IVs, etc.
  - used to verify that the key exchange and authentication was successful

Finished messages…

- Contains the MD5 and SHA-1 hash of all the previous handshake messages:
  - MD5(master_secret | pad_2 | MD5(handshake_messages | sender | master_secret | pad_1)) |
  - SHA(master_secret | pad_2 | SHA(handshake_messages | sender | master_secret | pad_1))
    - where "sender" is a code that identifies that the sender is the client or the server (client: 0x434C4E54; server: 0x53525652)

Cryptographic computations

- Pre-master secret
  - if key exchange is RSA based:
- generated by the client
- sent to the server encrypted with the server's public RSA key
  - if key exchange is Diffie-Hellman based:
- pre_master_secret = g^{xy} mod p

key block :

client write MAC secret	server write MAC secret	client write key	server write key	···

Master Secret (48 bytes)

master_secret = MD5(pre_master_secret | SHA(“A” |
pre_master_secret | client_random | server_random)) |
MD5(pre_master_secret | SHA(“BB” |
pre_master_secret | client_random | server_random)) |
MD5(pre_master_secret | SHA(“CCC” |
pre_master_secret | client_random | server_random))

Keys, MAC secrets, IVs

MD5(master_secret | SHA(“A” | master_secret | client_random | server_random)) |

MD5(master_secret | SHA(“BB” | master_secret | client_random | server_random)) |

MD5(master_secret | SHA(“CCC” | master_secret | client_random | server_random)) | …

Key Exchange Alternatives-
RSA / no client authentication

- server sends its encryption capable RSA public key in server_certificate
- server_key_exchange is not sent
- client sends encrypted pre-master secret in client_key_exchange
- client_certificate and certificate_verify are not sent

RSA/no client authentication

OR

- server sends its RSA or DSS public signature key in server_certificate
- server sends a temporary RSA public key in server_key_exchange
- client sends encrypted pre-master secret in client_key_exchange
- client_certificate and certificate_verify are not sent

Key exchange alternatives- RSA / client is authenticated

- server sends its encryption capable RSA public key in server_certificate
- server_key_exchange is not sent
- client sends its RSA or DSS public signature key in client_certificate
- client sends encrypted pre-master secret in client_key_exchange
- client sends signature on all previous handshake messages in certificate_verify

RSA / client is authenticated

OR

- server sends its RSA or DSS public signature key in server_certificate
- server sends a one-time RSA public key in server_key_exchange
- client sends its RSA or DSS public signature key in client_certificate
- client sends encrypted pre-master secret in client_key_exchange
- client sends signature on all previous handshake messages in certificate_verify

Key exchange alternatives Fix DH / no client authentication

- server sends its fix DH parameters in server_certificate
- server_key_exchange is not sent
- client sends its one-time DH public value in client_key_exchange
- client_ certificate and certificate_verify are not sent

Fix DH / client is authenticated

- server sends its fix DH parameters in server_certificate
- server_key_exchange is not sent
- client sends its fix DH parameters in client_certificate
- client_key_exchange is sent but empty
- certificate_verify is not sent

Key exchange alternatives- Ephemeral DH / no client auth

- server sends its RSA or DSS public signature key in server_certificate
- server sends signed one-time DH parameters in server_key_exchange
- client sends one-time DH public value in client_key_exchange
- client_certificate and certificate_verify are not sent

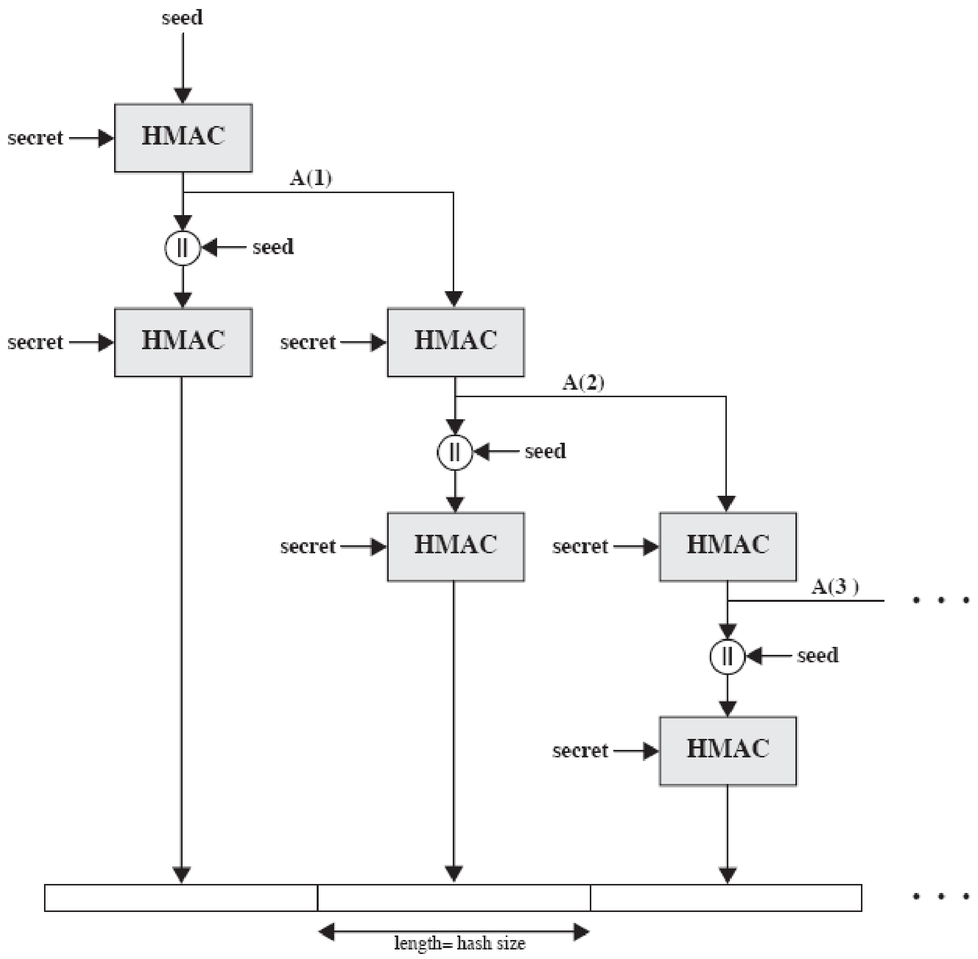
Ephemeral DH / client is authenticated

- server sends its RSA or DSS public signature key in server_certificate
- server sends signed one-time DH parameters in server_key_exchange
- client sends its RSA or DSS public signature key in client_certificate
- client sends one-time DH public value in client_key_exchange
- client sends signature on all previous handshake messages in certificate_verify

Key Exchange Alternatives- Anonymous DH/no Client Auth

- server_certificate is not sent
- server sends (unsigned) one-time DH parameters in server_key_exchange
- client sends one-time DH public value in client_key_exchange
- client_certificate and certificate_verify are not sent
- Anonymous DH / client is authenticated
  - not allowed

TLS Function P_hash(secret, seed)

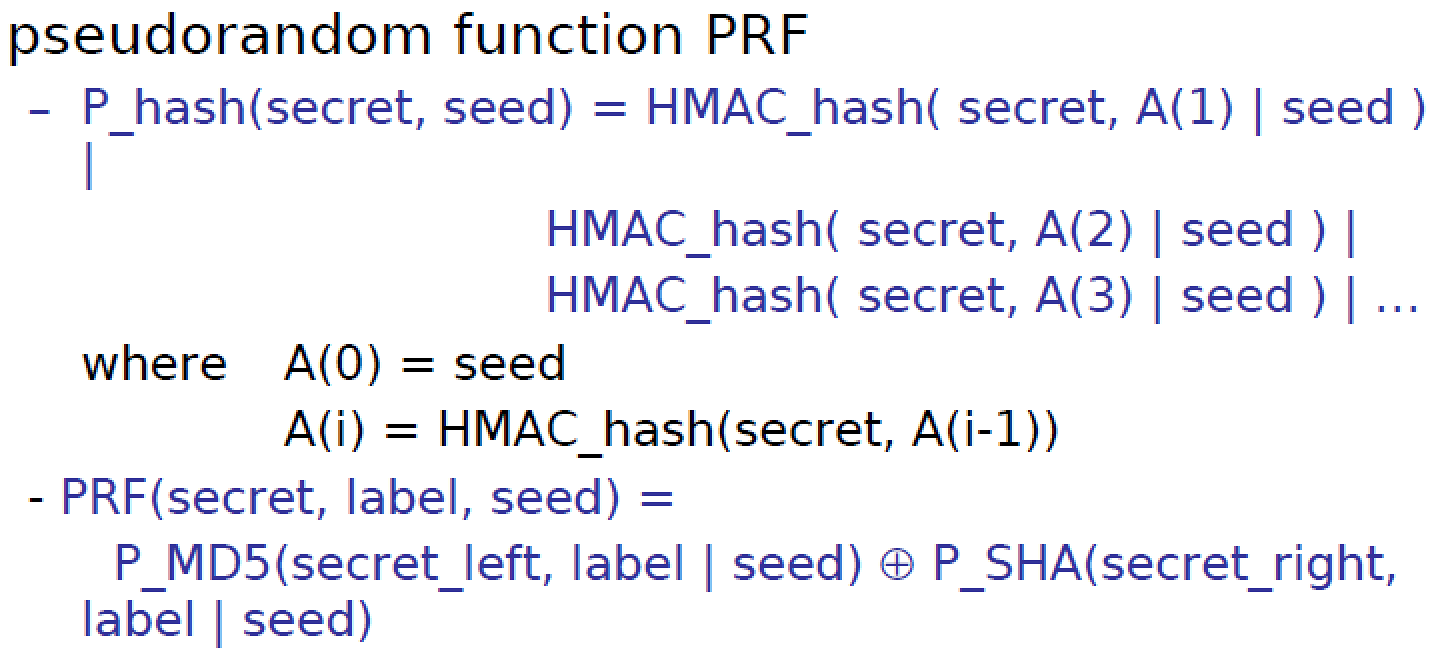


TLS vs. SSL

- version number
  - for TLS the current version number is 3.1
- MAC
  - TLS uses HMAC
  - the MAC covers the version field of the record header too
- more alert codes
- cipher suites
  - TLS doesn’t support Fortezza key exchange and Fortezza encryption

TLS vs. SSL

- certificate_verify message
  - the hash is computed only over the handshake messages
  - in SSL, hash contained the master_secret and pads



pseudorandom function PRF

- P_hash(secret, seed) = HMAC_hash(secret, A(1) | seed) |
  HMAC_hash(secret, A(2) | seed) |
  HMAC_hash(secret, A(3) | seed) | …

  where A(0) = seed
  A(i) = HMAC_hash(secret, A(i-1))
- PRF(secret, label, seed) =
  P_MD5(secret_left, label | seed) ⊕ P_SHA(secret_right, label | seed)

TLS vs. SSL

- finished message

```
PRF( master_secret,
     "client finished",
      MD5(handshake_messages) | SHA(handshake_messages) )
```

- cryptographic computations
  - pre-master secret is calculated in the same way as in SSL
  - master secret:

```
PRF( pre_master_secret,
     "master secret",
      client_random | server_random )
```

  - key block:

```
PRF( master_secret,
     "key expansion",
      server_random | client_random )
```

TLS vs. SSL

- Padding before block cipher encryption
  - variable length padding is allowed (max 255 padding bytes)

Any question ?