Ασφάλεια στο επίπεδο μεταφοράς

Νικόλαος Ε. Κολοκοτρώνης
Επίκουρος Καθηγητής

Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Πανεπιστήμιο Πελοποννήσου
Email: nkolok@uop.gr
Web: http://www.uop.gr/~nkolok/

ΑΣΦΑΛΕΙΑ ΣΥΣΤΗΜΑΤΩΝ

Περιεχόμενα

- Secure Sockets Layer (SSL)
  - Επιθέσεις
- Transport Layer Security (TLS)
- HTTP πάνω από SSL/TSL
- Secure Shell (SSH)

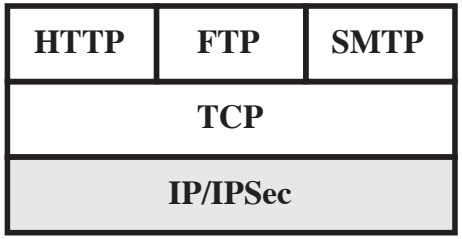
Ασφάλεια στο web

- The Web is fundamentally a client/server application
- Tailored security due to
  - Web servers are relatively easy to configure and manage
  - Web content is increasingly easy to develop
  - The underlying software is extraordinarily complex
  - May hide many potential security flaws
- A Web server can be exploited to gain access to entire computer net
- Many users untrained in security matters
  - they are not necessarily aware of the security risks that exist
  - they don’t have the tools/ knowledge to take effective countermeasures

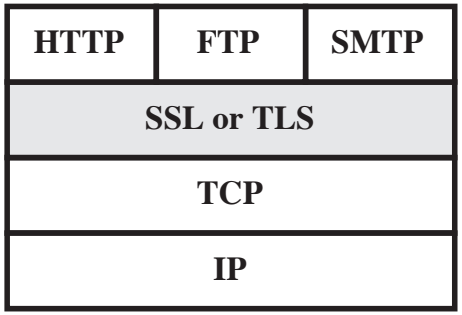
Απειλές στο web

	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

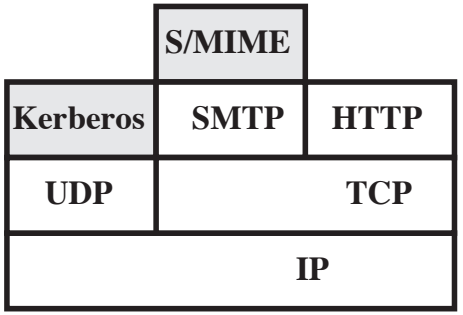
Μηχανισμοί ασφάλειας



(a) Network Level



(b) Transport Level



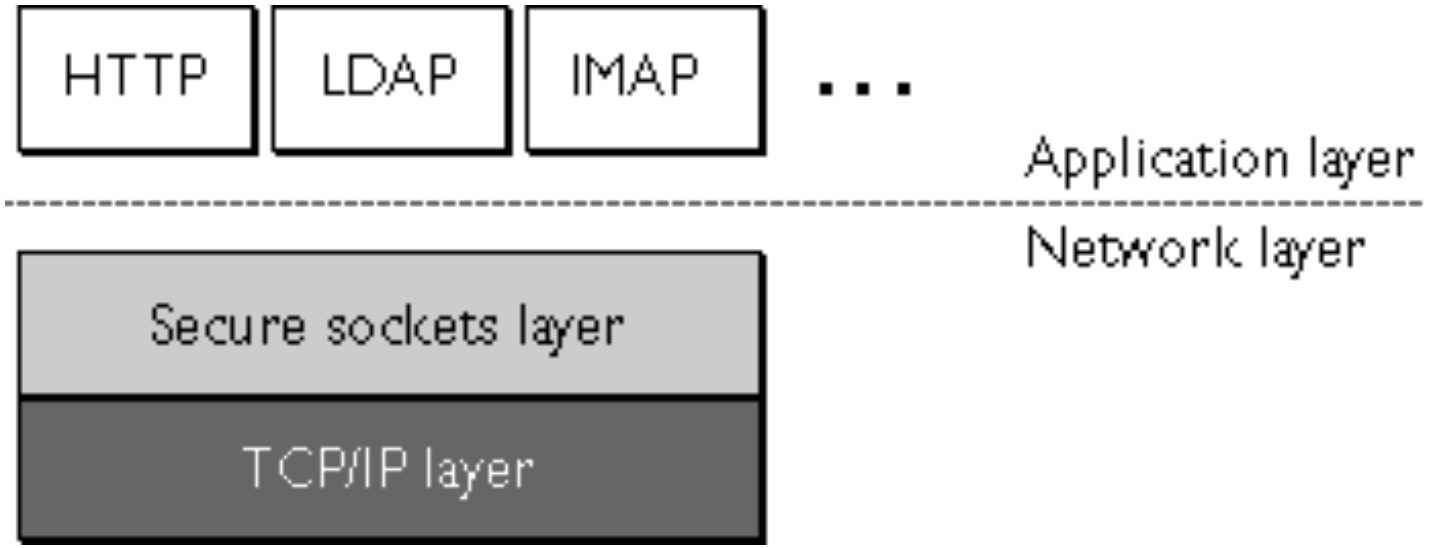
(c) Application Level

Πρωτόκολλο SSL

Secure sockets layer

- One of the most widely used security services
- A general purpose service implemented as a set of protocols that rely on TCP
  - Could be provided as part of the underlying protocol suite and therefore be transparent to applications
  - Can be embedded in specific packages

Secure sockets layer



- Runs on top of layer 4
- TCP provides reliable transmission of packets

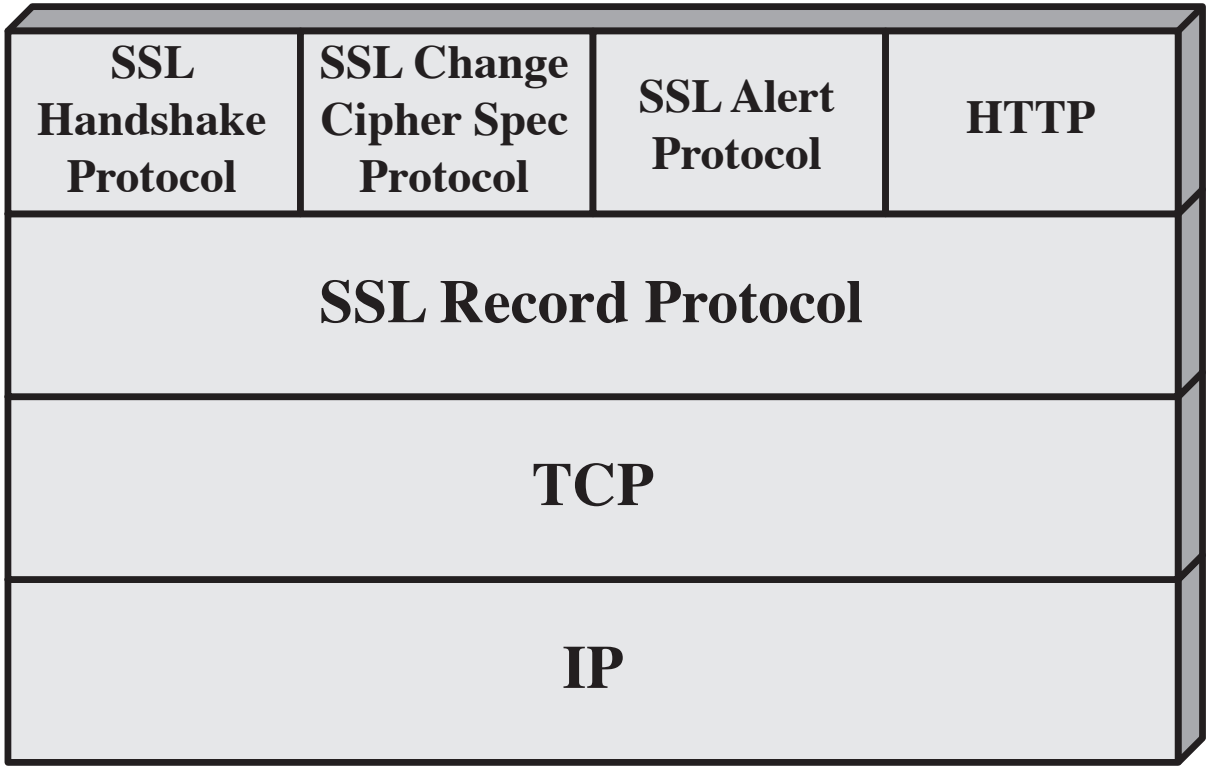
Secure sockets layer

- SSL was developed by Netscape Communications Inc. for transmitting private documents via the Internet
- The primary goal of SSL is to provide privacy and reliability between two communicating applications
- SSL is built into all major browsers and web servers (broadly adopted, implemented, used)
- Both Netscape Navigator and Internet Explorer support SSL, and many websites use the protocol to obtain confidential user information, such as credit card numbers

Secure sockets layer

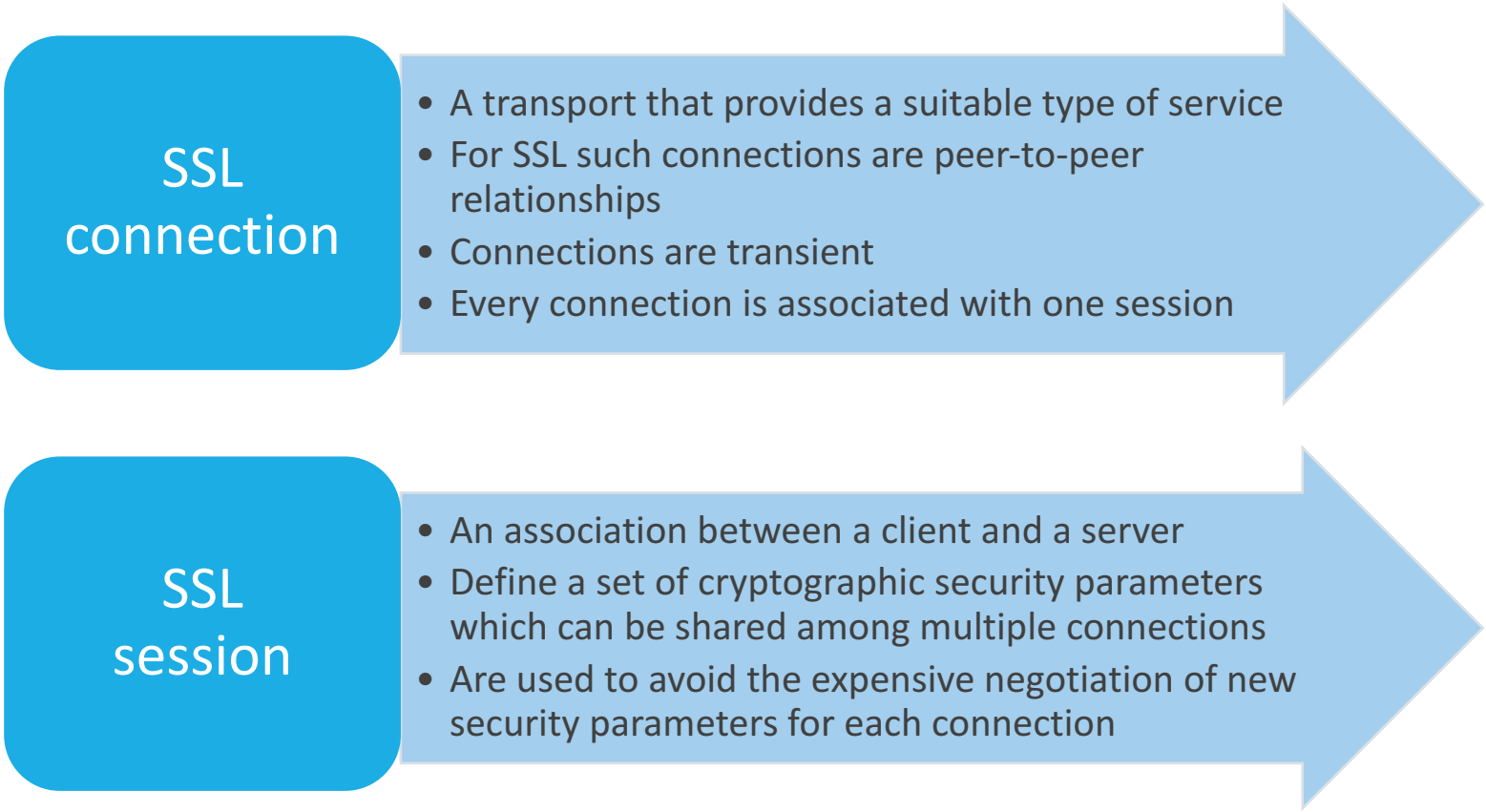
- 1994 (mid): SSL v1.0 - first version
- 1994 (end): SSL v2.0 - first product ships
- 1995 (beg): SSL v2.0 - reference implementation
- 1995 (end): SSL v3.0 - latest version
  - Reduce number of roundtrips
  - Support more key exchange and cipher algorithms
  - Re-negotiate ciphers from current spec
  - Separate authentication and encryption keys

SSL protocol stack

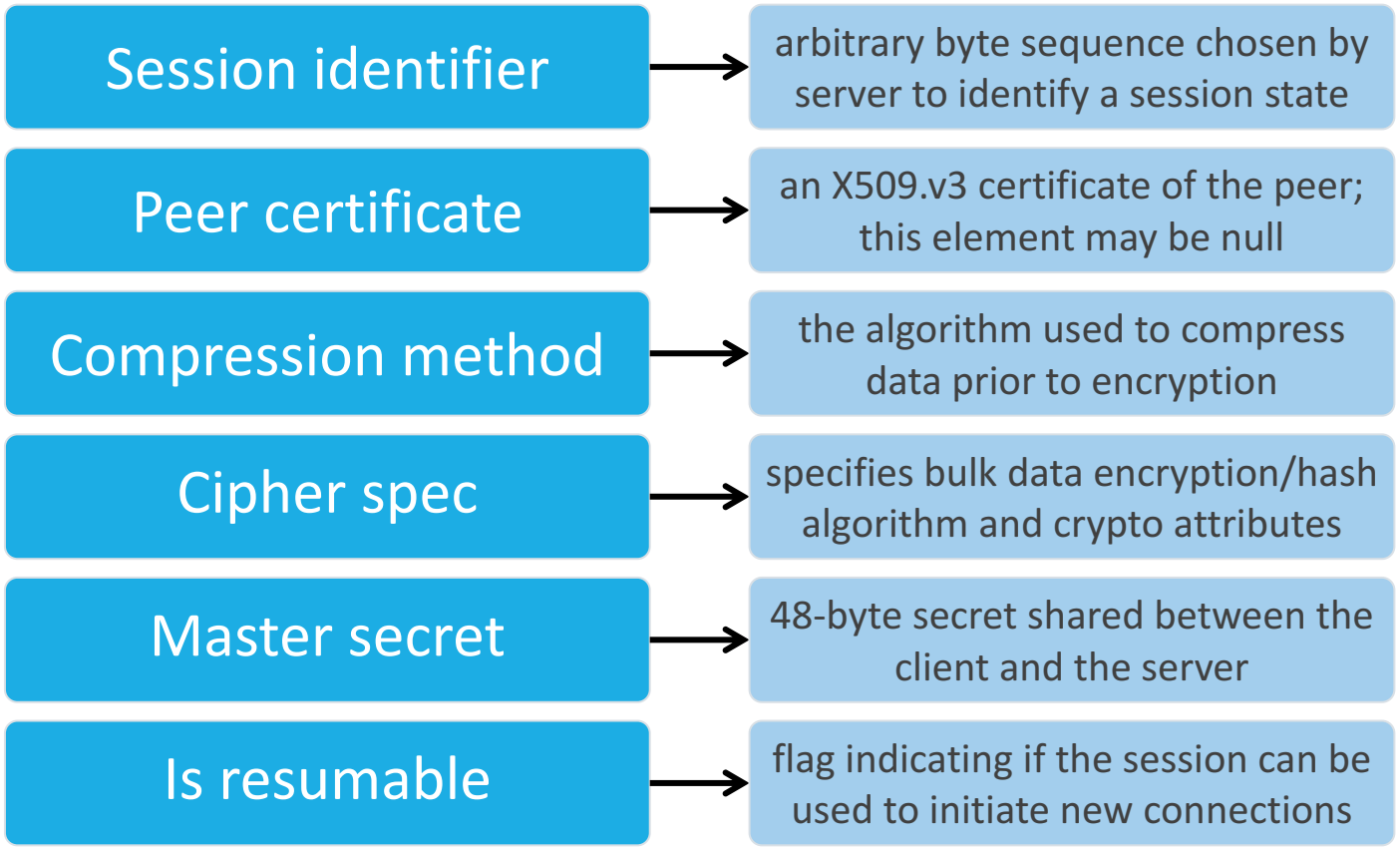


SSL architecture

- Two important SSL concepts are:



SSL session state

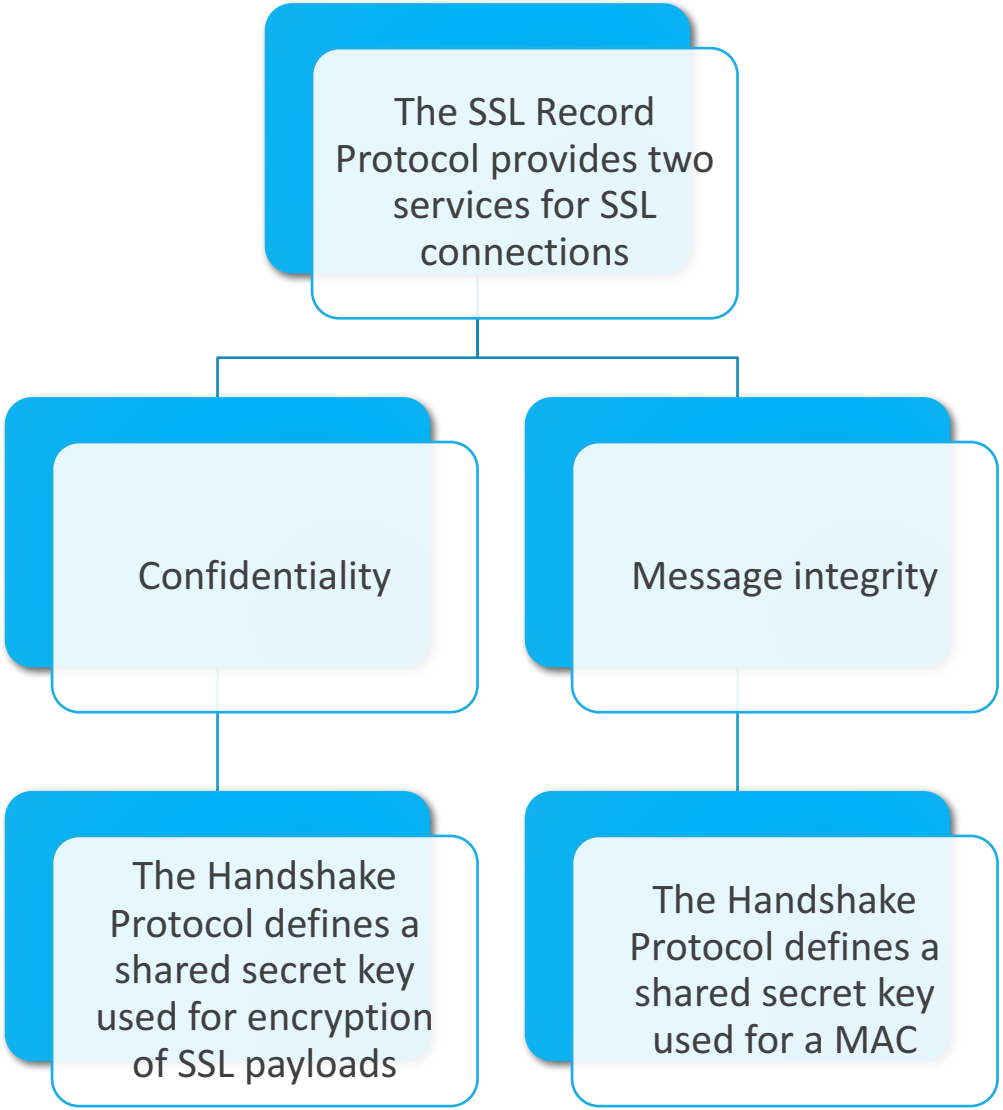


SSL connection state

Server and client random	• byte sequences chosen by the server and client
Server write MAC secret	• the secret key used in MAC by the server
Client write MAC secret	• the secret key used in MAC by the client
Server write key	• the symmetric encryption key used by the server
Client write key	• the symmetric encryption key used by the client

Initialization vectors	• initialized by SSL handshake (for CBC) • final ciphertext block of a record preserved for use as the IV with the next record
sequence numbers	• kept for transmitted/ received messages • is set to 0 if sending or receiving a change cipher spec message • numbers may not exceed $2^{64} - 1$

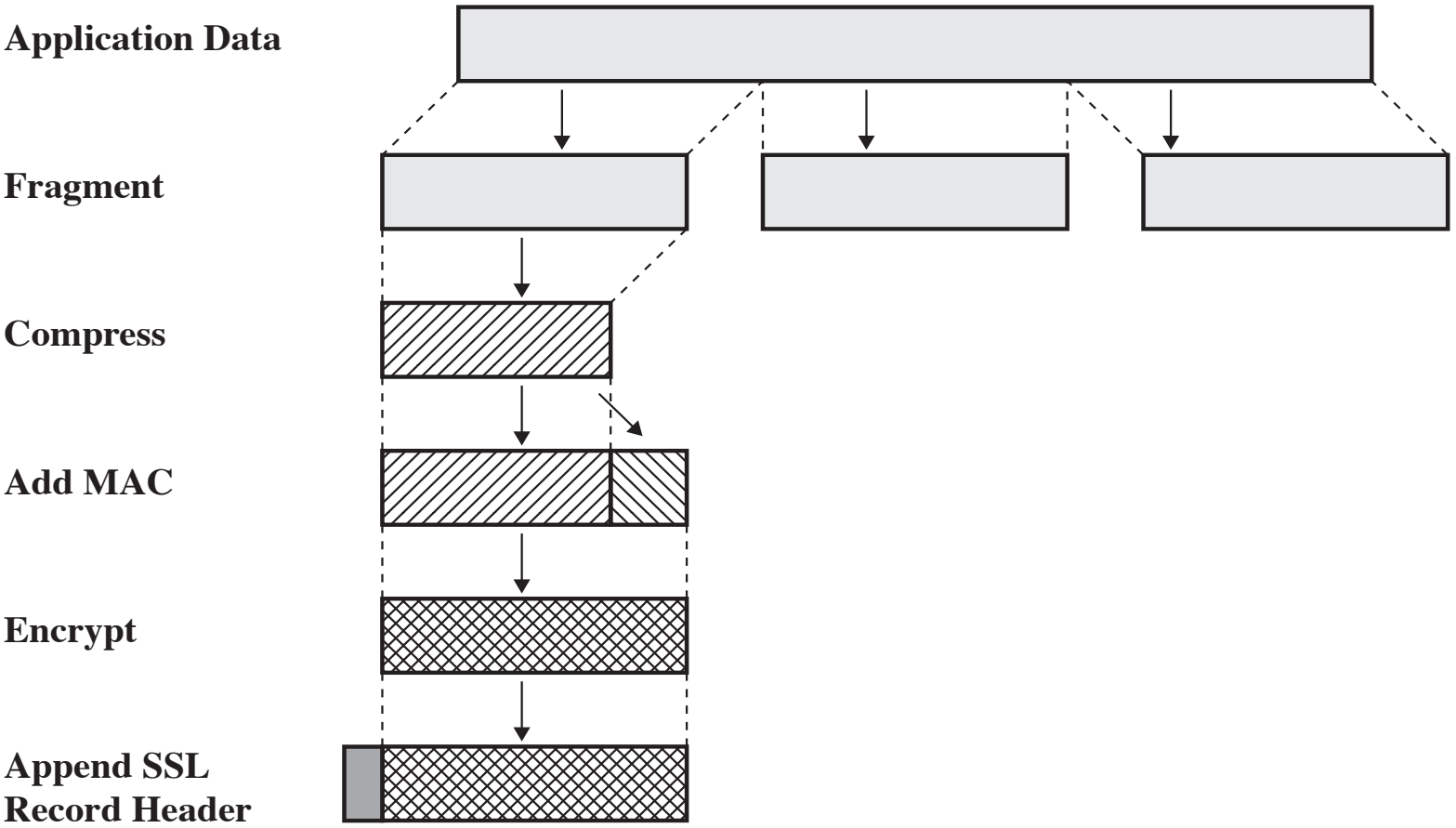
SSL record protocol



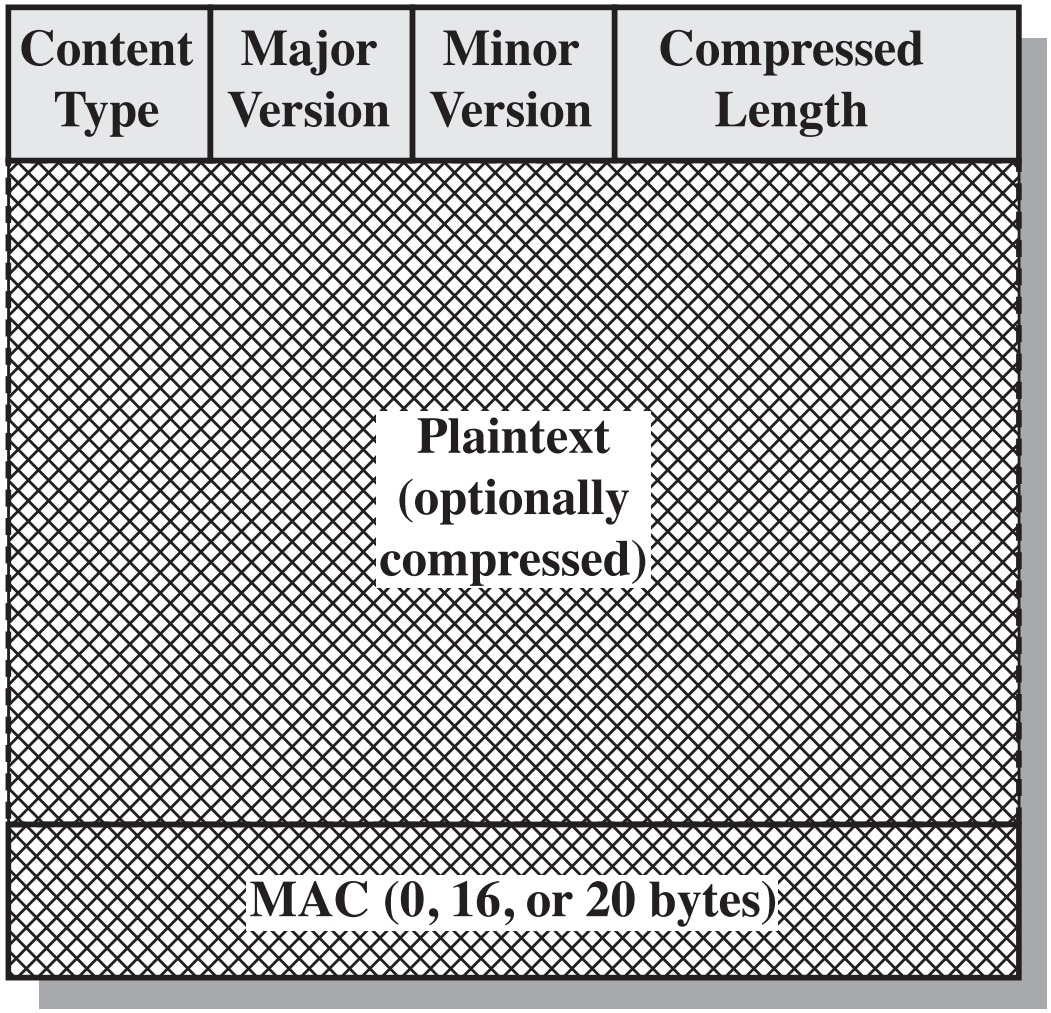
SSL record protocol

- Fragmentation
- Compression (optional)
- Message authentication code (MAC)
- Encryption
- Header addition

SSL record protocol operation



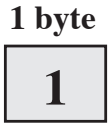
SSL record format



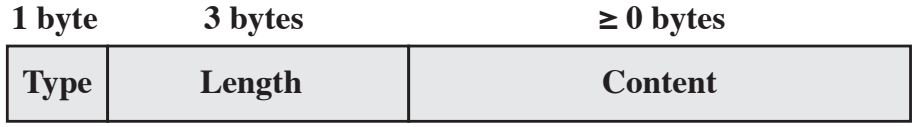
SSL record format

- Content type
  - Application data
  - Alert
  - Handshake
  - Change Cipher Spec
- Content length
  - Suggests when to start processing
- SSL version
  - Redundant check for version agreement

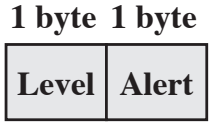
SSL record protocol payload



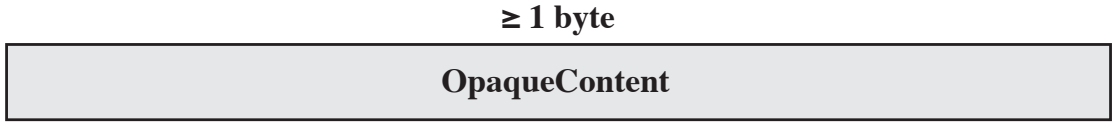
(a) Change Cipher Spec Protocol



(c) Handshake Protocol



(b) Alert Protocol



(d) Other Upper-Layer Protocol (e.g., HTTP)

SSL change cipher spec

- Signals transitions in ciphering strategies
  - Consists of a single message, which is encrypted and compressed under the current Cipher Spec
  - The message consists of a single byte of value 1
- The message is sent by both the client and server
  - Notify each other that subsequent records will be protected under the just-negotiated CipherSpec and keys

SSL alert

- It is used to convey SSL-related alerts to the peer entity
  - Different levels of alerts (fatal, non-fatal)
  - Alert messages are encrypted and compressed, as specified by the current connection state
- Fatal alerts result in immediate termination of connection
  - Other connections corresponding to the session may continue
  - However, the session identifier must be cancel, preventing the failed session from being used to establish new connections

SSL alert

- Closure alerts
  - Share knowledge that connection is ending in order to avoid attacks
  - Truncation attack
    - sending a TCP FIN before the sender is finished

SSL error types

- Fatal errors
  - unexpected_message
  - bad_record_mac
  - decompression_failure
  - handshake_failure
    - Unable to negotiate an acceptable set of security parameters given the options available
  - illegal_parameter

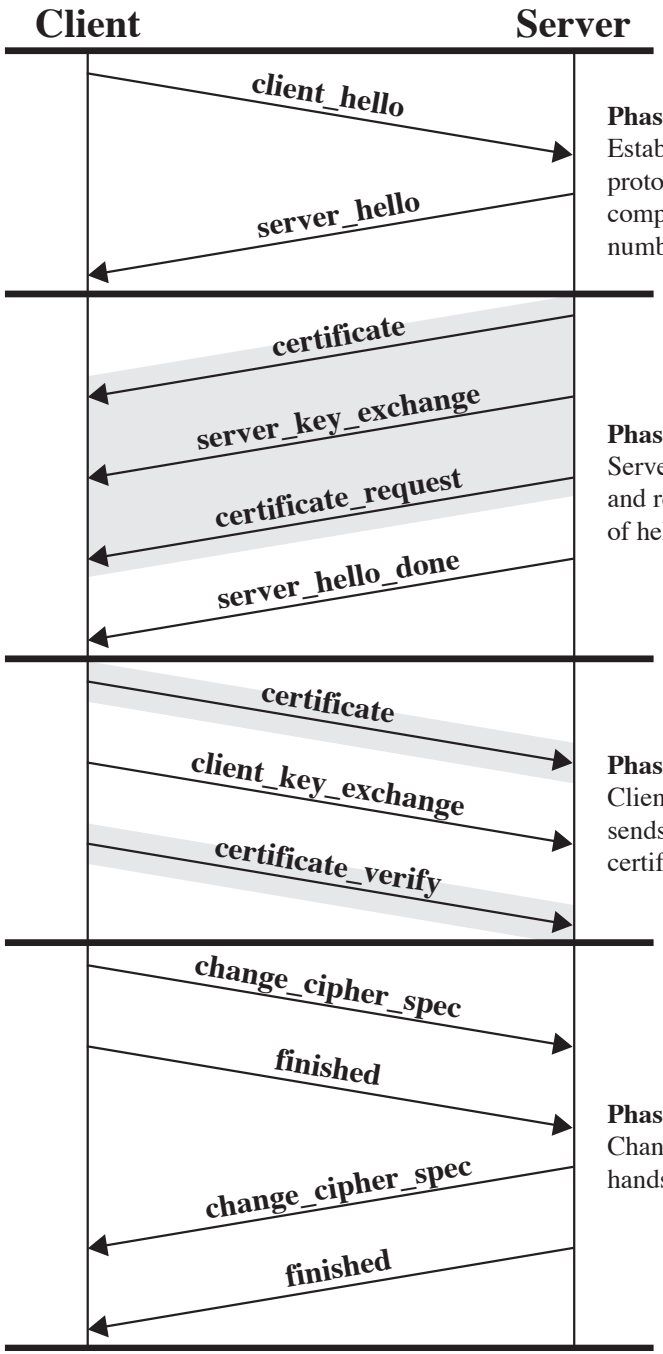
SSL error types

- Non-fatal errors
  - certificate_revoked
  - certificate_expired
  - unsupported_certificate (e.g. of incorrect type)
  - no_certificate (declares non-availability of a certificate)
  - bad_certificate (due to corruption, etc.)
  - certificate_unknown

SSL handshake message types

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

SSL handshake



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

Phase 4
Change cipher suite and finish handshake protocol.

Cryptographic computations

- Two further items are of interest:
  - The creation of a shared master secret by means of the key exchange
    - The shared master secret is a one-time 48-byte value generated for this session by means of secure key exchange
- The generation of cryptographic parameters from the master secret

Cryptographic computations

- CipherSpec requires the following, which are generated from the master secret in that order
  - client write MAC secret
  - server write MAC secret
  - client write key
  - server write key
  - client write IV
  - server write IV

SSL client_hello

- Protocol version
  - SSL v3.0 (major=3, minor=0)
- Random Number
  - 32 bytes – first 4 bytes, time of day in sec, other 28 bytes random
  - Prevents replay attack
- Session ID
  - 32 bytes – indicates the use of previous cryptographic material
- Compression algorithm

SSL list of algorithms

- SSL_{ASYMMETRIC}_WITH_{SYMMETRIC}_{HASH}
  - SSL_RSA_WITH_RC4_128_MD5 = { 0, 4 }
  - SSL_RSA_WITH_RC4_128_SHA = { 0, 5 }
  - SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 = { 0, 6 }
  - SSL_RSA_WITH_IDEA_CBC_SHA = { 0, 7 }
  - SSL_RSA_EXPORT_WITH_DES40_CBC_SHA = { 0, 8 }
  - SSL_RSA_WITH_DES_CBC_SHA = { 0, 9 }
  - SSL_RSA_WITH_3DES_EDE_CBC_SHA = { 0, 10 }

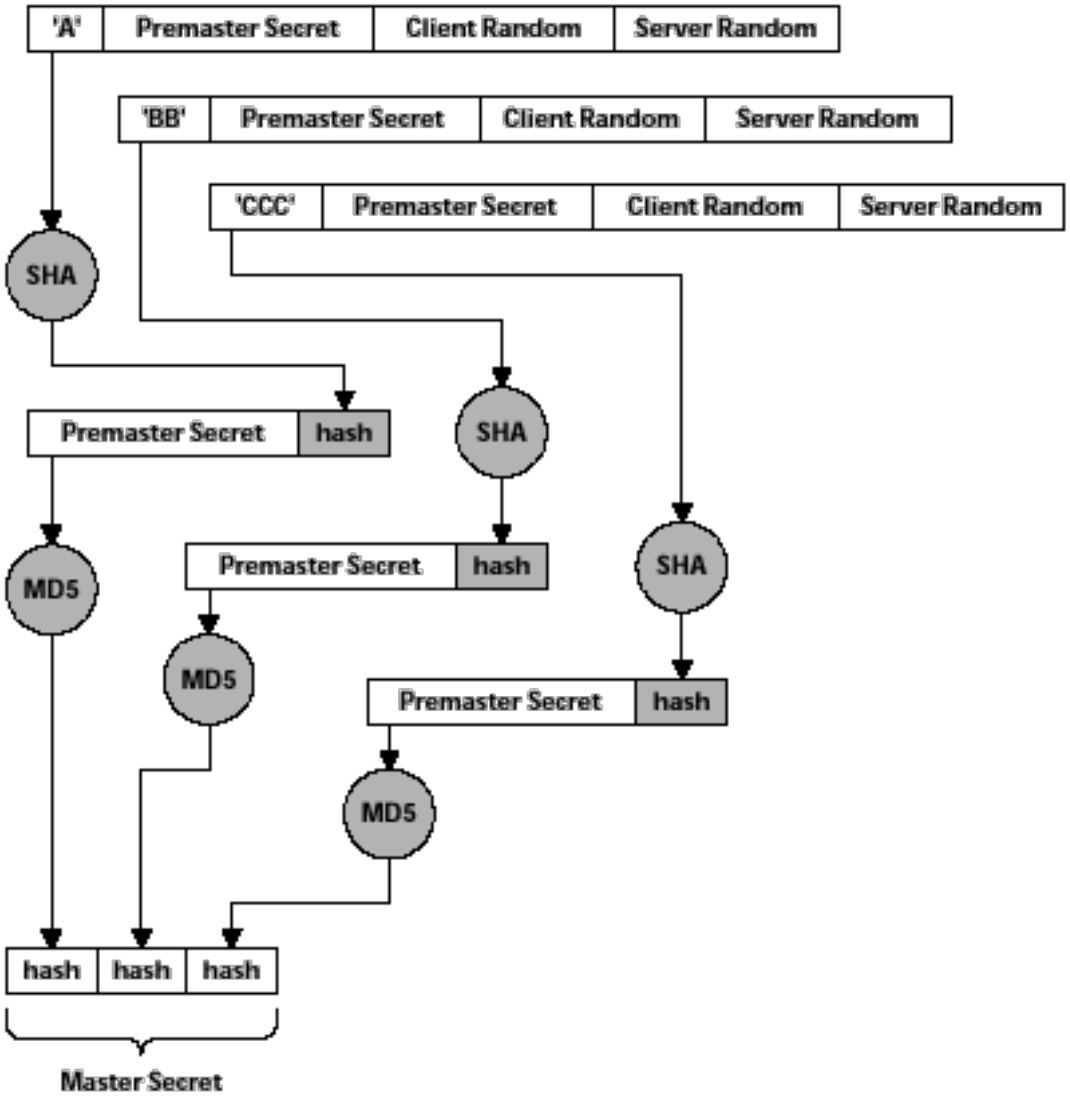
SSL server_hello

- Version
- Random Number
  - Protects against handshake replay
- Session ID
  - Provided to the client for later resumption of the session
- Cipher suite
  - Usually picks client's best preference
- Compression method

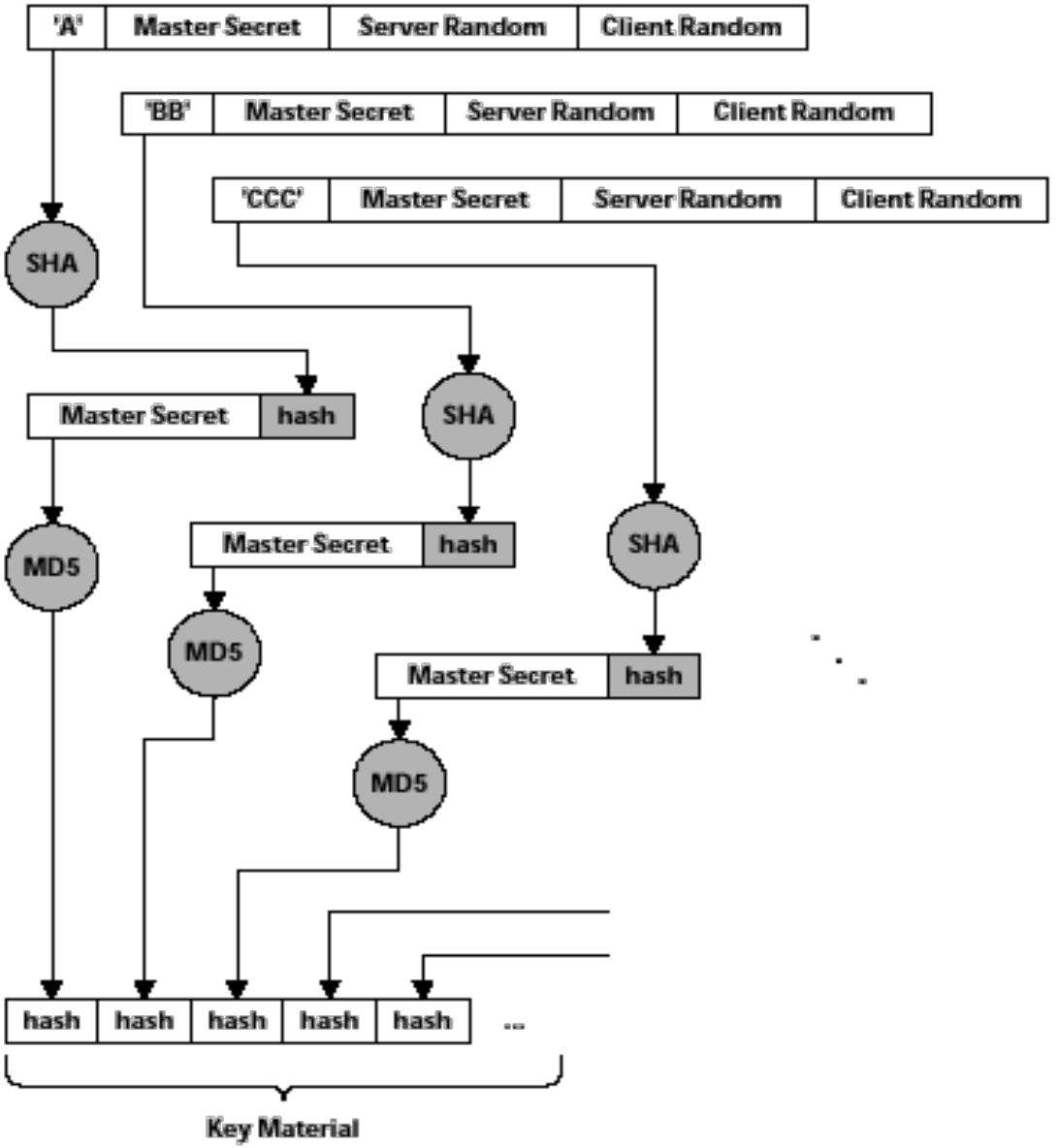
Key material generation

- Premaster secret
  - Created by client; used to “seed” calculation of encryption params
  - 2 bytes of SSL version + 46 random bytes
  - Sent encrypted to server using server’s public key
- Master secret
  - Generated by both parties from premaster secret and random values generated by both client and server
- Key material
  - Generated from the master secret and shared random values
- Encryption keys are extracted from the key material

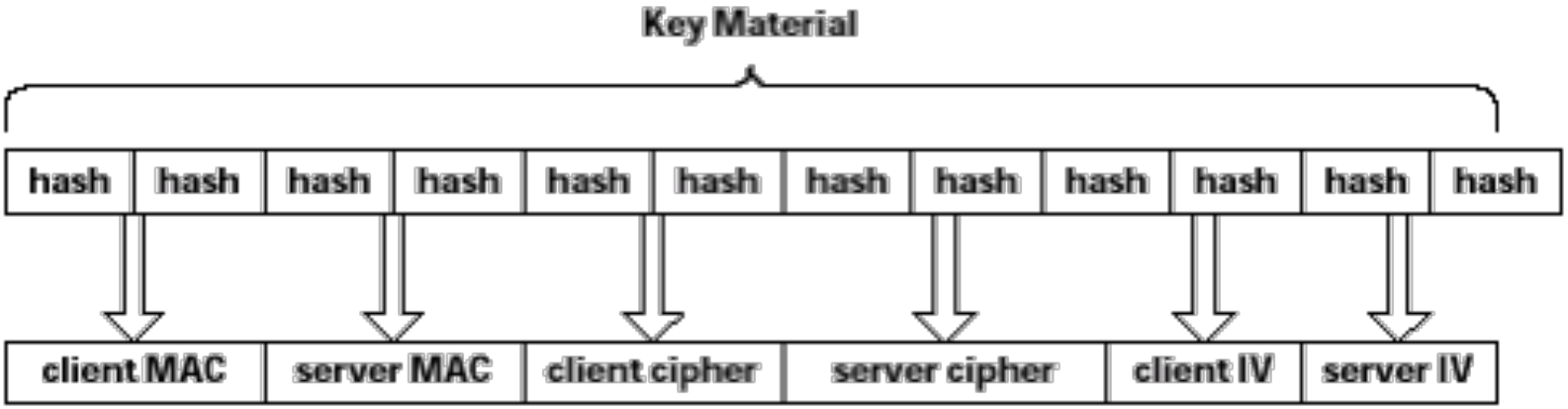
SSL master secret



SSL key material



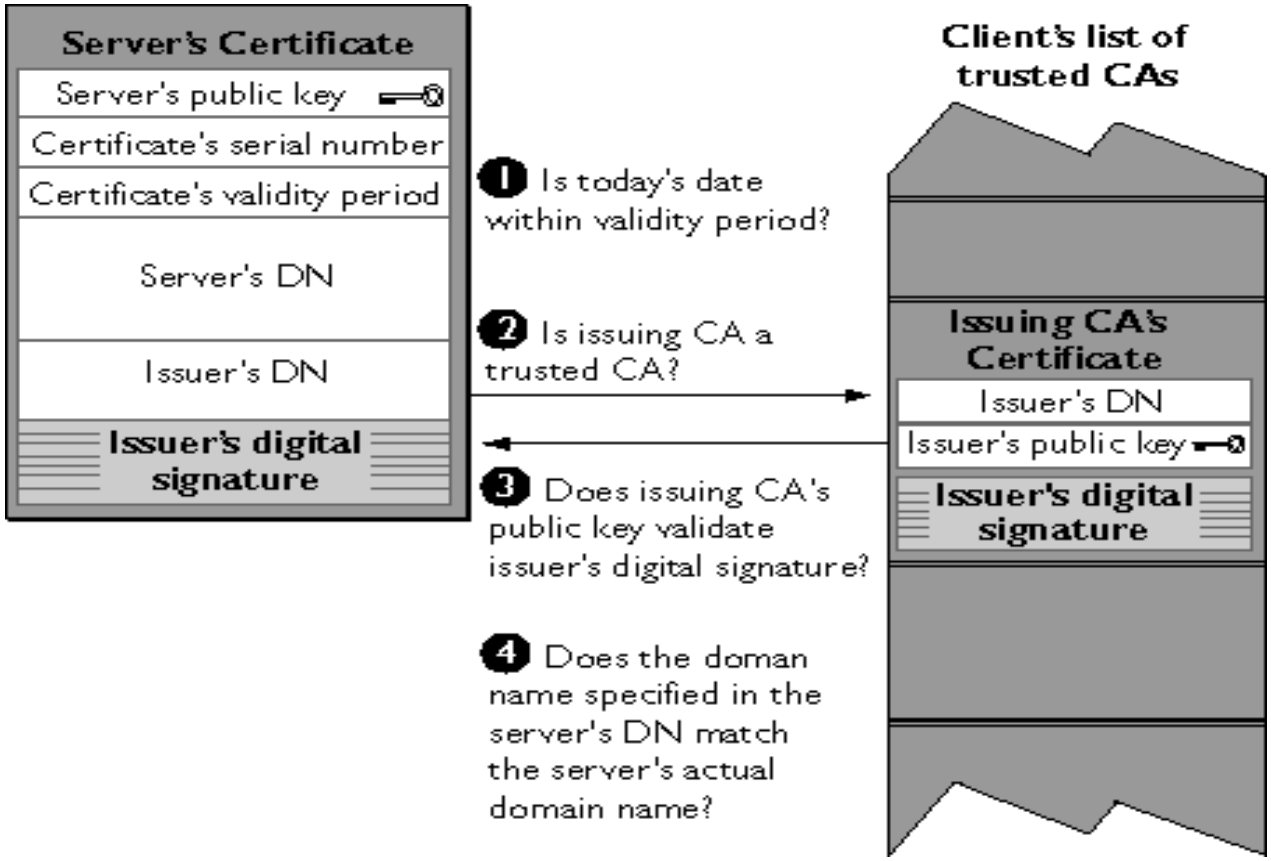
SSL key material



SSL certificates

- SSL clients such as Browser stores some CA's public keys
- User of the client can add or delete CA's public keys
- SSL servers need to get public key certificates issued by CAs
- When SSL server sends its certificate to a SSL client, the client can verify it
- Client certificates and authentication are not supported widely

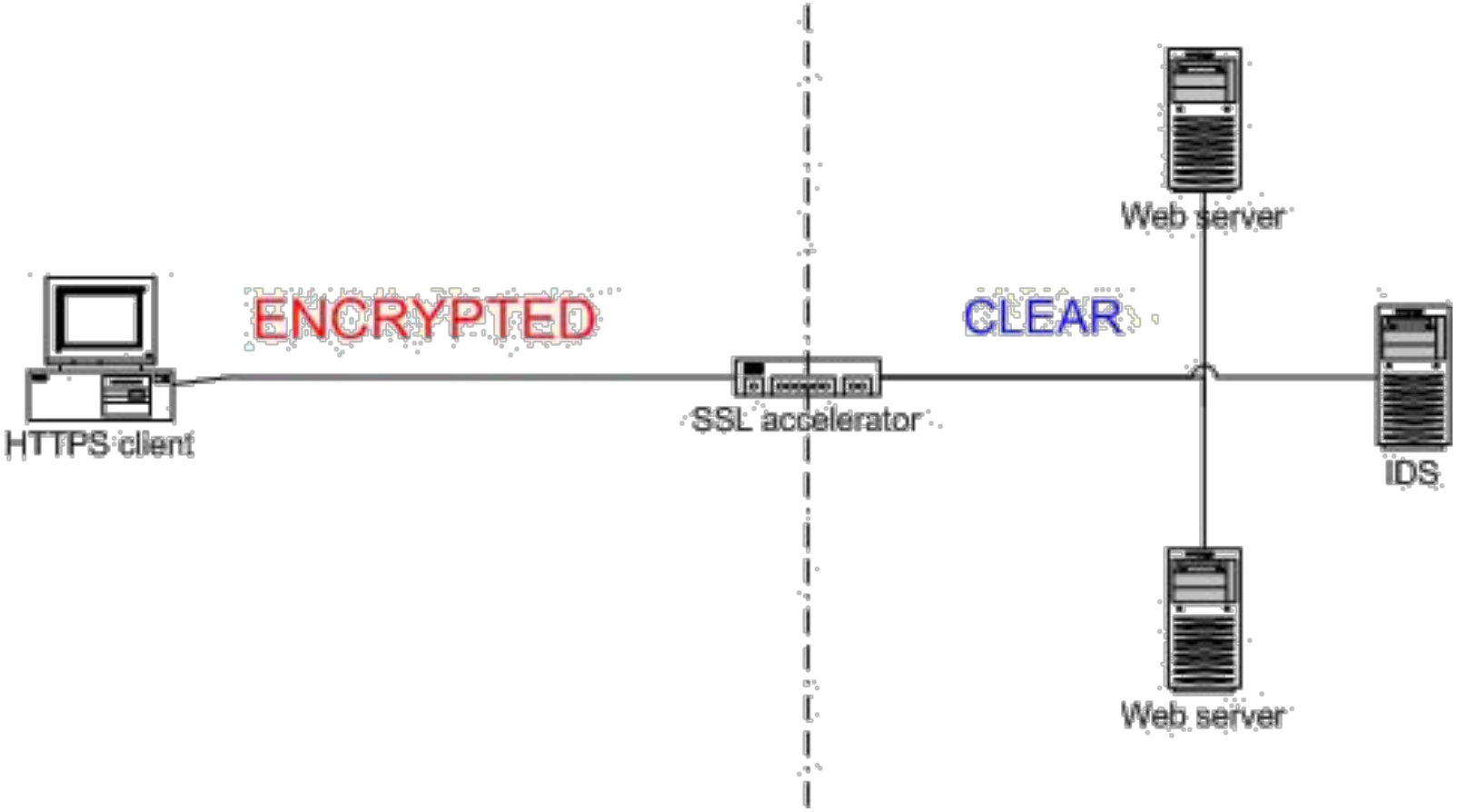
SSL certificate verification



SSL performance

- Degradation of 50% is sometimes cited compared with sending in the clear (2-10 times slower than a TCP session)
- Result of public key encryption and decryption required to initialize session (handshake phase)
  - Client does public-key encryption
  - Server does private-key encryption (still public-key cryptography)
  - Usually clients have to wait on servers to finish
- Overhead of encryption using RCx/DES is practically noise
  - Symmetric key encryption imposes less overhead

SSL performance



Επιθέσεις SSL

Το χρονικό των επιθέσεων

1996: Δημοσιεύεται το SSL 3.0, από τη Netscape

1999: Η IETF δημοσιεύει το TLS 1.0

2001: Ο Vaudenay παρατηρεί την ύπαρξη Μαντείων Συμπληρώματος στο SSL 3.0 και TLS 1.0

2002: Ο Wei Dai δημοσιεύει μία επίθεση εναντίον του SSH2 με προβλέψιμα διανύσματα αρχικοποίησης (IVs)

Το χρονικό των επιθέσεων

2002: Ο Bodo Moller παρατηρεί ότι η επίθεση του Wei Dai ισχύει και για το SSL 3.0

2006: Δημοσιεύεται το TLS 1.1, προσπάθεια επιδιόρθωσης

2008: Δημοσιεύεται το TLS 1.2, με νέους AEAD αλγορίθμους κρυπτογράφησης

2011: Οι Thai και Duong δημοσιεύουν το BEAST

2014: Οι Moller και Duong δημοσιεύουν το POODLE

Η δομή του CBC στο SSL 3.0

```
struct {
   opaque IV[SecurityParameters.record_iv_length];
   block-ciphered struct {
      opaque content[SSLCompressed.length];
      opaque MAC[SecurityParameters.mac_length];
      uint8 padding[GenericBlockCipher.padding_length];
      uint8 padding_length;
   };
} GenericBlockCipher;
```

Προβλήματα χρήσης του MAC

- ✘ MAC then Encrypt (SSL)
  - Η MAC κρυπτογραφείται μαζί με το κείμενο
  - Αποστέλλεται το τελικό κρυπτογράφημα
- ✘ MAC and Encrypt (SSH)
  - Η MAC παράγεται πάνω στο κείμενο
  - Αποστέλλεται μαζί με το κρυπτογράφημα
- ☑ Encrypt then MAC (IPSec)
  - Η MAC παράγεται πάνω στο κρυπτογράφημα
  - Αποστέλλεται μαζί με το κρυπτογράφημα

Επίθεση POODLE

- Αρχικά: Padding Oracle On Downgraded Encryption
- Στο SSL 3.0 το padding
  - Παίρνει τυχαίες τιμές
  - Δεν καλύπτεται από τη MAC
  - Μόνο το τελευταίο byte υποδηλώνει το μήκος
- Αδύνατον για το server να ξεχωρίσει αν έχει γίνει πλαστογράφηση
  - Το SSL δεν πρέπει να χρησιμοποιείται πλέον

Πως προχωράμε

- Όταν σχεδιάστηκε το SSL 3.0 δεν ήταν ξεκάθαρο ότι η τεχνική *encrypt-then-MAC* ήταν η πιο ασφαλής
  - Ο ελλιπής έλεγχος του στο συμπλήρωμα ήταν καταδικαστικός
  - Ο CBC έχει ελαττώματα, όμως έχουμε εναλλακτικές
- Οι επιθέσεις γίνονται μόνο καλύτερες, συνδυάζονται, και γίνονται πιο σύνθετες
- Στο TLS 1.3 γίνεται προσπάθεια για να αποφευχθούν τα λάθη του παρελθόντος

Πρωτόκολλο TLS

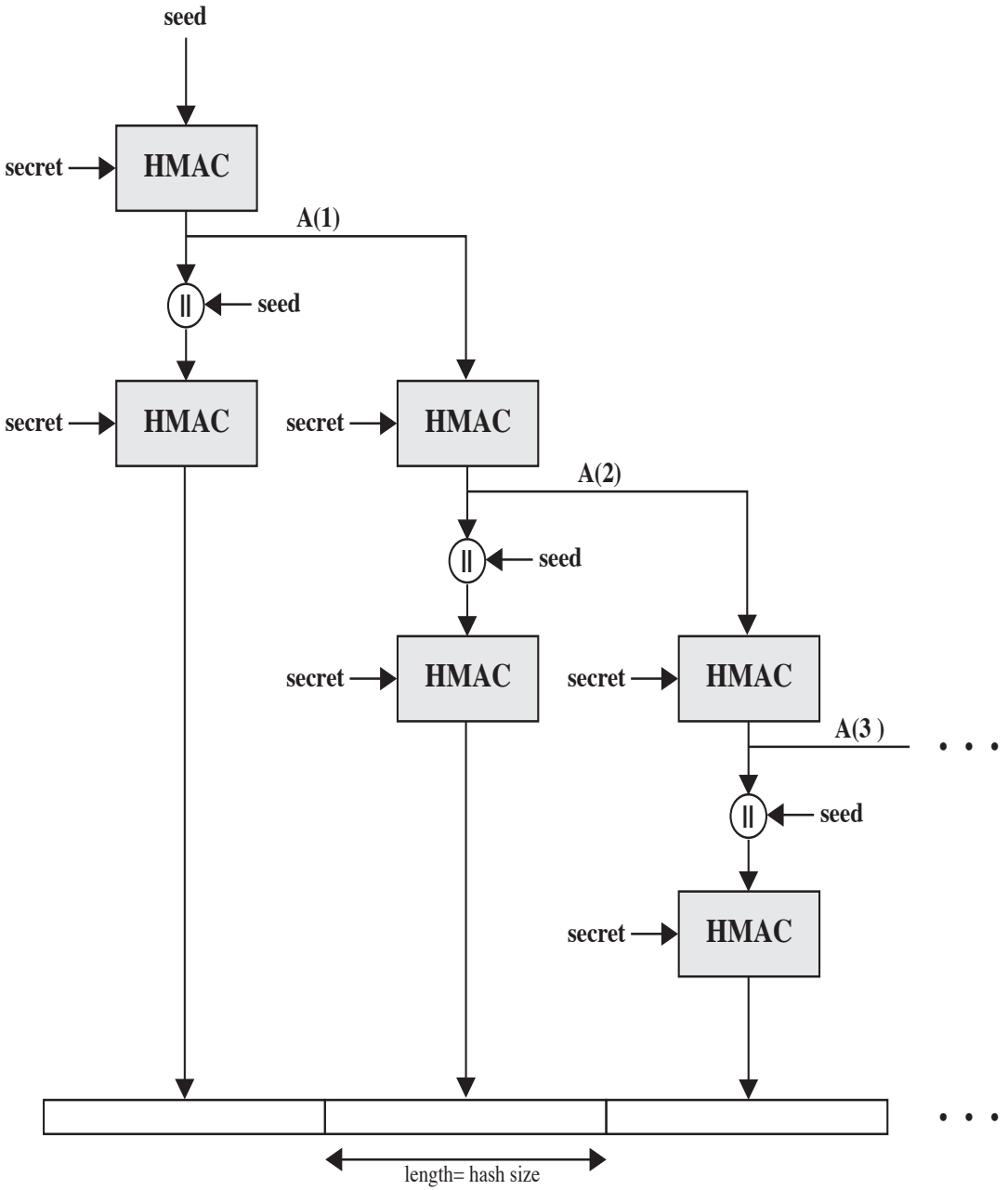
Transport layer security

- An IETF standardization initiative whose goal is to produce an Internet standard version of SSL
- Is defined as a Proposed Internet Standard in RFC 5246
  - RFC 5246 is very similar to SSLv3

Differences include

- Version number
- Message authentication code
- Pseudorandom function
- Alert keys
- Cipher suites
- Client certificate types
- Certificate_verify and finished Messages
- Cryptographic computations
- Padding

TLS function



Πρωτόκολλο HTTPS

HTTPS (HTTP over SSL/TLS)

- Combination of HTTP and SSL/TLS to implement secure communication between web browsers and web servers
  - The HTTPS capability is built into all modern Web browsers
- A user of a Web browser will see URL addresses that begin with https:// rather than http://
- If HTTPS is specified, port 443 is used, which invokes SSL
- Documented in RFC 2818, *HTTP Over TLS*
  - There is no fundamental change in using HTTP over either SSL or TLS and both implementations are referred to as HTTPS

HTTPS (HTTP over SSL/TLS)

- When HTTPS is used, the following elements of the communication are encrypted:
  - URL of the requested document
  - Contents of the document
  - Contents of browser forms
  - Cookies sent from browser to server and from server to browser
  - Contents of HTTP header

Connection initiation

- For HTTPS, the agent acting as the HTTP client also acts as the TLS client
  - The client initiates a connection to the server on the appropriate port
    - Then, sends the TLS ClientHello to begin the TLS handshake
  - When the TLS handshake has finished, the client may then initiate the first HTTP request
  - All HTTP data is to be sent as TLS application data

Connection initiation

- Three levels of awareness of a connection in HTTPS
  - At the HTTP level, an HTTP client requests a connection to an HTTP server by sending a connection request to the next lowest layer
    - Typically the next lowest layer is TCP, but it may also be TLS/SSL
  - At the level of TLS, a session is established between a TLS client and a TLS server
    - This session can support one or more connections at any time
  - A TLS request to establish a connection begins with the establishment of a TCP connection between the TCP entity on the client side and the TCP entity on the server side

Connection closure

- An HTTP host indicates the closing of a connection by including `Connection: close` in an HTTP record
- The closure of an HTTPS connection requires that TLS closes the connection with the peer TLS entity
  - involves closing the underlying TCP connection
- TLS implementations must initiate an exchange of closure alerts before closing a connection
  - A TLS implementation may, after sending a closure alert, close the connection without waiting for the peer to send its closure alert
  - Unannounced TCP closures could imply existence of an attack

Κι άλλες εφαρμογές

- SSL-protected HTTP (https on default port 443)
- SSL-protected SMTP <mail sending> (ssmtp on port 465)
- SSL-protected Usenet News (snews on port 563)
- SSL-protected LDAP (ssl-ldap on port 636)
- SSL-protected POP3 <mail retrieval> (spop3 on port 995)

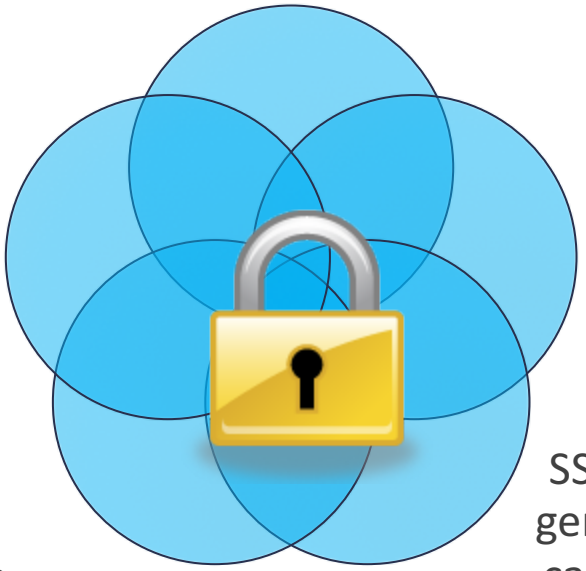
Πρωτόκολλο SSH

Secure shell

Protocol for secure network comm. that is simple to implement

SSH in most OSs

- for remote login and X tunneling
- one of the most pervasive apps



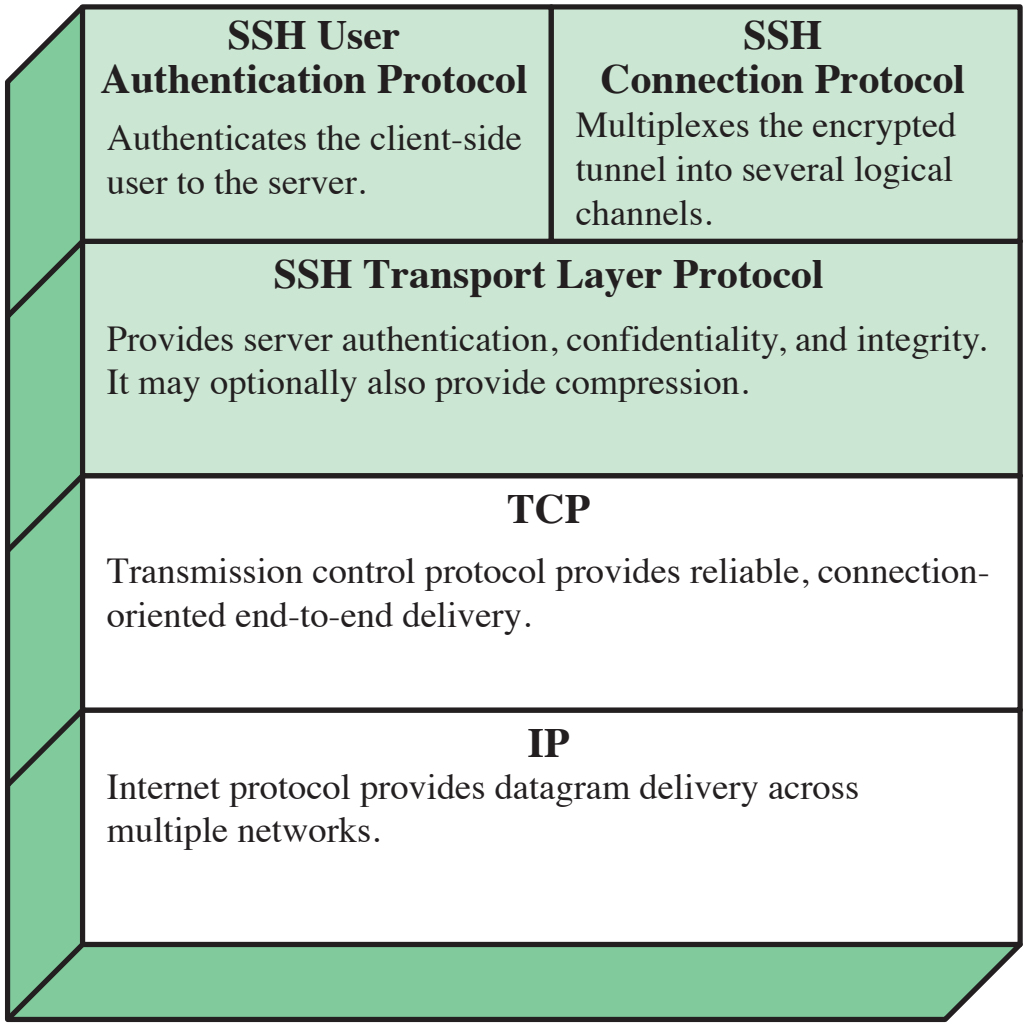
SSH1 was built to replace remote logon schemes with no security

SSH2 fixes some security flaws

- in IETF RFCs 4250 - 4256

SSH provides a more general capability and can be used for more network functions

SSH protocol stack



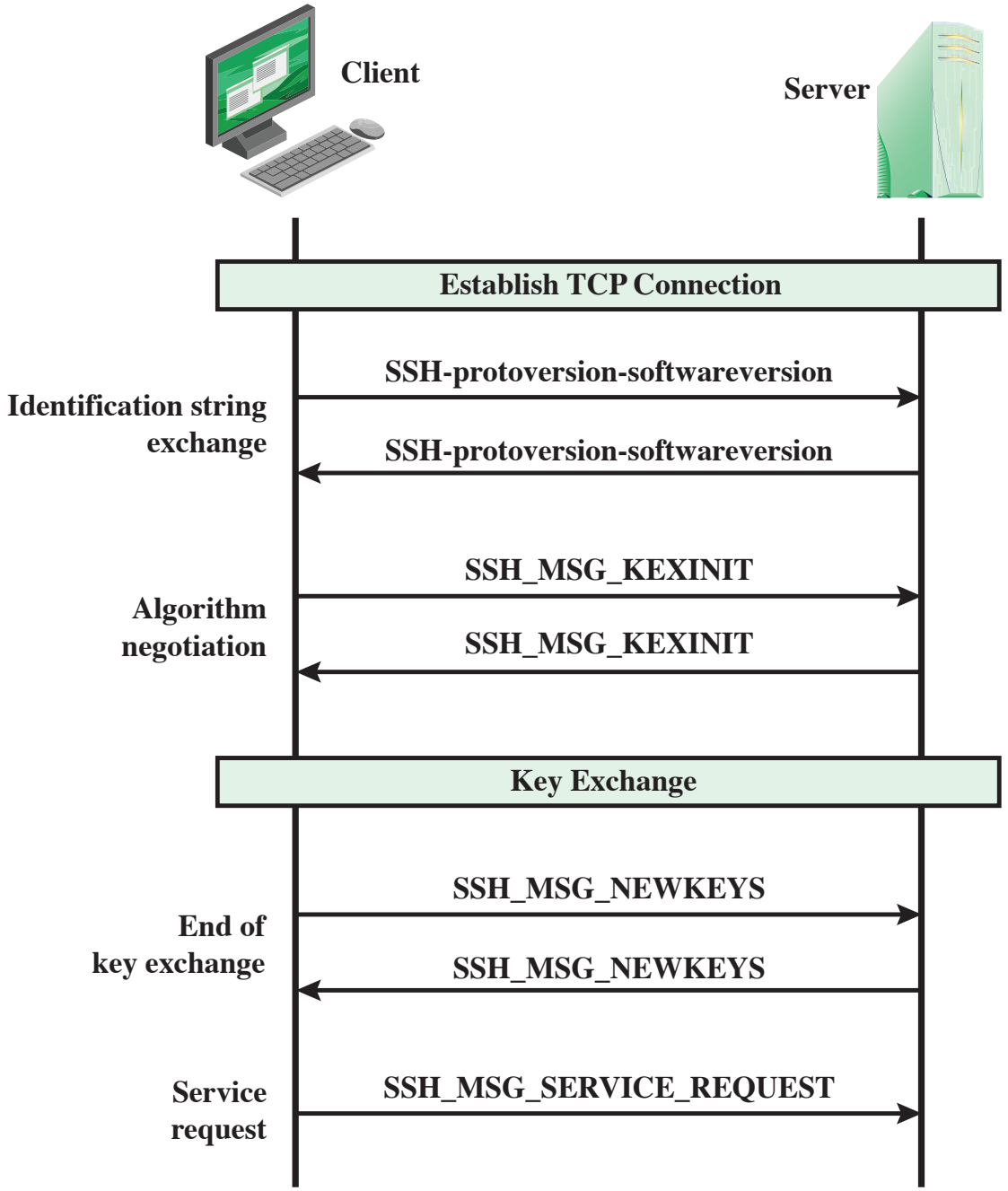
Transport layer protocol

- Server authentication occurs at the transport layer, based on the server possessing a public/private key pair
- A server may have multiple host keys using multiple different asymmetric encryption algorithms
- Multiple hosts may share the same host key
- The server host key is used during key exchange to authenticate the identity of the host

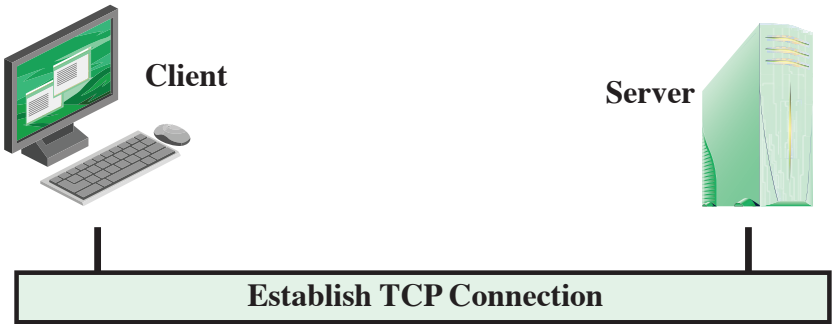
Transport layer protocol

- RFC 4251 dictates two alternative trust models:
  - The client has a local database that associates each host name with the corresponding public host key
  - The host name-to-key association is certified by a trusted certification authority (CA);
    - the client only knows the CA root key and can verify the validity of all host keys certified by accepted CAs

SSH TLP packet exchanges

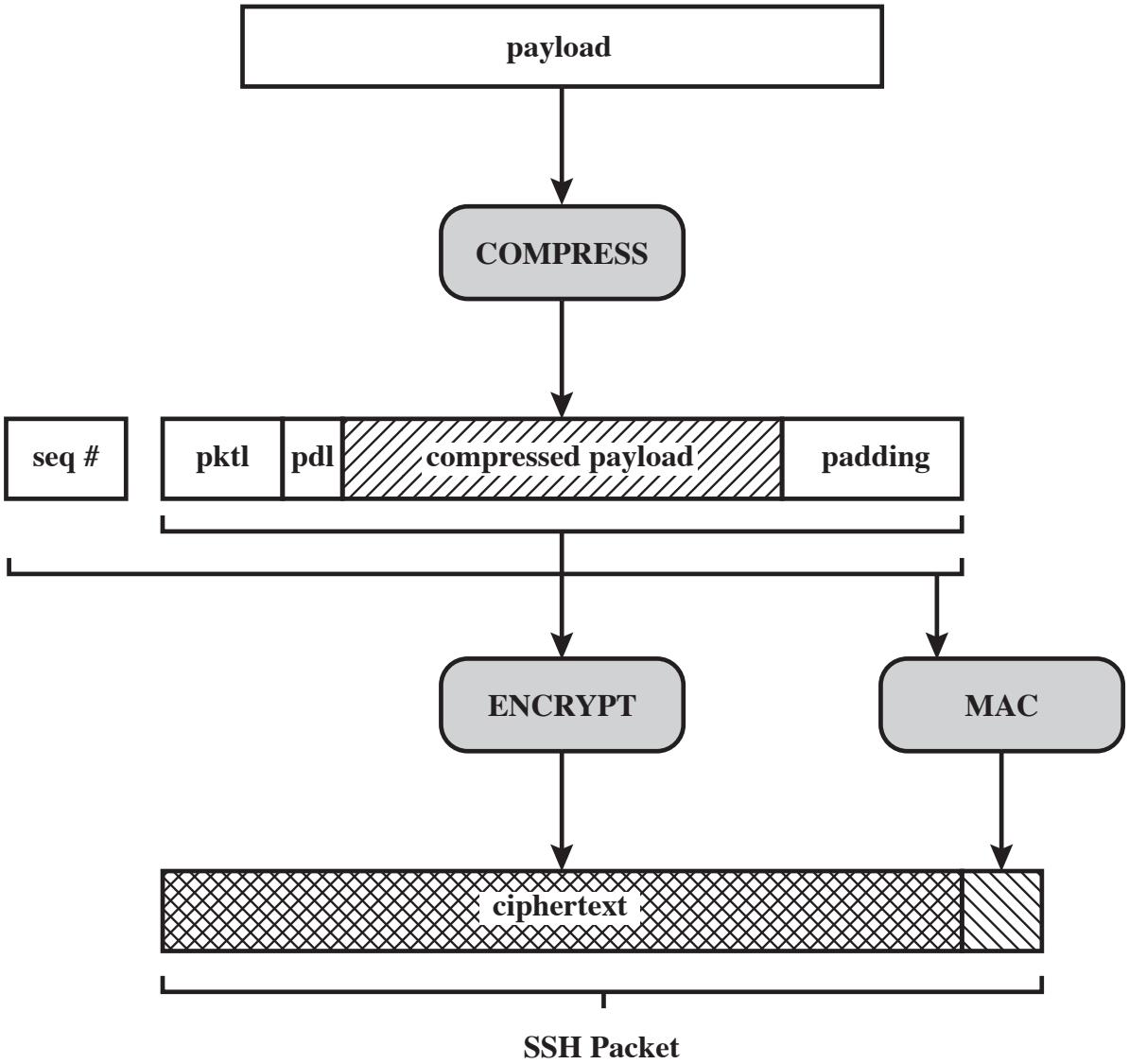


Transport layer protocol



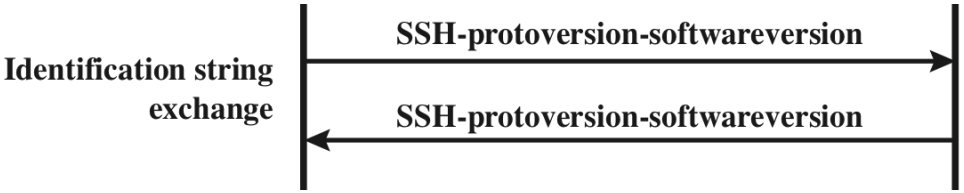
- The client initiates the connection
- The server listens on TCP port 22

SSH TLP packet formation



pktl = packet length
pdl = padding length

Transport layer protocol

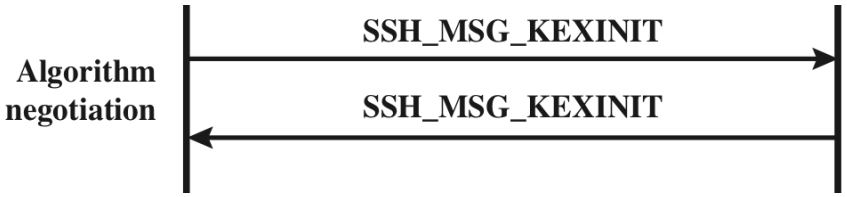


- Both side must send a version string of the following form:

  `SSH-protoversion-softwareversion SP comments CR LF`

- SP, CR, LF are space character, carriage return, line feed
- Used to indicate the capabilities of an implementation
- Following packets are sent with the Binary Packet Protocol

Transport layer protocol



- Each side sends an SSH_MSG_KEXINIT
  - contains lists of supported algs in the order of preference
  - assume these messages to be I_C and I_S
- There is one list for each type of crypto alg.
  - types = key exchange, encryption, MAC, compression

Transport layer protocol

- Δομή μηνύματος SSH_MSG_KEXINIT
  - kex_algorithms (comma separated list of names)
  - server_host_key_algorithms
  - encryption_algorithms_client_to_server (and _server_to_client)
  - mac_algorithms_client_to_server (and _server_to_client)
  - compression_algorithms_client_to_server (and _server_to_client)
  - first_kex_packet_follows (boolean)
  - random cookie (16 bytes)

Supported algorithms

Cipher	
3des-cbc*	Three-key 3DES in CBC mode
blowfish-cbc	Blowfish in CBC mode
twofish256-cbc	Twofish in CBC mode with a 256-bit key
twofish192-cbc	Twofish with a 192-bit key
twofish128-cbc	Twofish with a 128-bit key
aes256-cbc	AES in CBC mode with a 256-bit key
aes192-cbc	AES with a 192-bit key
aes128-cbc**	AES with a 128-bit key
Serpent256-cbc	Serpent in CBC mode with a 256-bit key
Serpent192-cbc	Serpent with a 192-bit key
Serpent128-cbc	Serpent with a 128-bit key
arcfour	RC4 with a 128-bit key
cast128-cbc	CAST-128 in CBC mode

MAC algorithm	
hmac-sha1*	HMAC-SHA1; digest length = key length = 20
hmac-sha1-96**	First 96 bits of HMAC-SHA1; digest length = 12; key length = 20
hmac-md5	HMAC-MD5; digest length = key length = 16
hmac-md5-96	First 96 bits of HMAC-MD5; digest length = 12; key length = 16

Compression algorithm	
none*	No compression
zlib	Defined in RFC 1950 and RFC 1951

* = Required
** = Recommended

Transport layer protocol

- Lists of algorithms
  - The server list the algorithms it supports
  - The client lists the algorithms that it is willing to accept
  - Algorithms are listed in order of preference
- Selection: first algorithm on the client's list that is also on the server's list

Transport layer protocol

- Any key exchange algorithm produces two values
  - A shared master key *K*
  - An exchange hash *H*

Transport layer protocol

- Currently two versions of DH key exchange are specified
  - C computes $e = g^x \bmod p$ for random x
  - S computes $f = g^y \bmod p$ for random y
  - Both compute $K = e^y \bmod p = f^x \bmod p$
- Both DH key exchange versions are defined in RFC 2409

Transport layer protocol

- *H* from the first key exchange is used as the session ID

$$H = \text{hash}(V_C \,||\, V_S \,||\, I_C \,||\, I_S \,||\, K_S \,||\, e \,||\, f \,||\, K)$$

- V_X is X's identification string
- I_X is X's SSH_MSG_KEXINIT message
- K_S is S's public host key

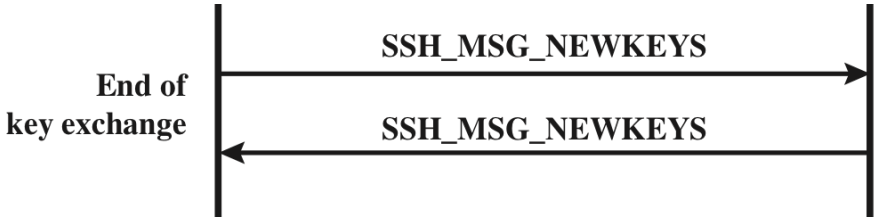
Transport layer protocol

- Keys and IVs are derived from K and H as follows
  - IV client to server = HASH(K | H | “A” | session ID)
  - IV server to client = HASH(K | H | “B” | session ID)
  - encryption key client to server = HASH(K | H | “C” | session ID)
  - encryption key server to client = HASH(K | H | “D” | session ID)
  - MAC key client to server = HASH(K | H | “E” | session ID)
  - MAC key server to client = HASH(K | H | “F” | session ID)
- where HASH is the hash function specified during the negotiation

Transport layer protocol

- If the key length is longer than the output of HASH…
  - K1 = HASH(K | H | X | session ID)
  - K2 = HASH(K | H | K1)
  - K3 = HASH(K | H | K1 | K2)
  - …
  - key = K1 | K2 | K3 | …

Transport layer protocol

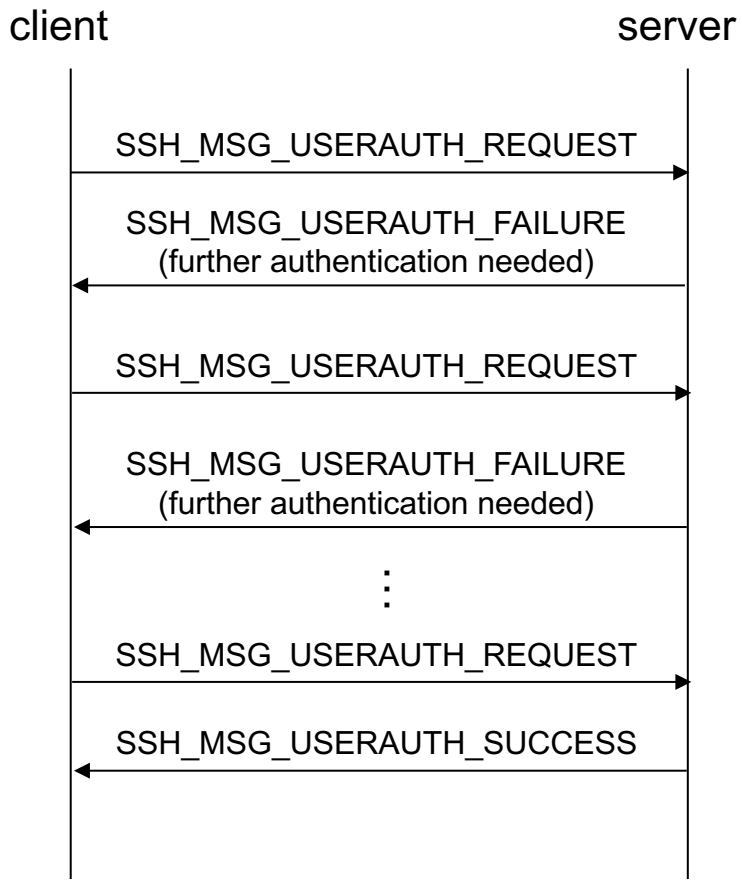


- The end of key exchange is signaled by the exchange of SSH_MSG_NEWKEYS packets
- At this point, both sides may start using the keys generated from K, as discussed subsequently

Transport layer protocol

- The client sends an SSH_MSG_SERVICE_ REQUEST packet to request either
  - the User Authentication (SSH-USERAUTH) or
  - the Connection Protocol (SSH-CONNECTION)
- Subsequent to this, all data is exchanged as the payload of an SSH TLP are protected by encryption and MAC

User authentication



- USERAUTH_REQUEST
  - Three methods supported
- USERAUTH_FAILURE
  - List of authentication methods that remain
  - Partial success flag (T/F)
- USERAUTH_SUCCESS
  - Authentication is complete
  - S starts the requested service

User authentication: methods

Publickey

- The client sends a message to the server that contains the client's public key, with the message signed by the client's private key
- When the server receives this message, it checks whether the supplied key is acceptable for authentication and, if so, it checks whether the signature is correct

Password

- The client sends a message containing a plaintext password, which is protected by encryption by the Transport Layer Protocol

hostbased

- Authentication is performed on the client's host rather than the client itself
- The client sends a signature created with the private key of the client host
- Instead of verifying user's identity, the SSH server verifies the identity of client host

User authentication: publickey

- SSH_MSG_USERAUTH_REQUEST
  - User name
  - Service name
  - **“publickey”**
  - TRUE (a flag set to TRUE)
  - Public key algorithm name (e.g., ssh-dss)
  - Public key
  - Signature

User authentication: password

- SSH_MSG_USERAUTH_REQUEST
  - User name
  - Service name
  - **“password”**
  - FALSE (a flag set to FALSE)
  - Password (not encrypted)
- All implementations should support this method
- This method is likely the most widely used

User authentication: hostbased

- SSH_MSG_USERAUTH_REQUEST
  - User name
  - Service name
  - **“hostbased”**
  - Public key algorithm name
  - Public key and certificates for client host
  - Client host name
  - User name on client host
  - Signature

Connection protocol

- The SSH Connection Protocol runs on top of the SSH Transport Layer Protocol and assumes that a secure authentication connection is in use
- The secure authentication connection, referred to as a *tunnel,* is used by the Connection Protocol to multiplex a number of logical channels

Connection protocol: channels

- All types of communication using SSH are supported using separate channels
  - Either side may open a channel
  - For each channel, each side associates a unique channel number
- Channels are flow controlled using a window mechanism
- No data may be sent to a channel until a message is received to indicate that window space is available
- The life of a channel progresses through three stages: opening a channel, data transfer, and closing a channel

Channel types

Session

- The remote execution of a program (may be a shell, an app like file transfer or e-mail, a system command, or some built-in subsystem)
- Once a session channel is opened, next requests are used to start the remote program

X11

- Refers to the X Window System, a computer software system and network protocol that provides a graphical user interface (GUI) for networked computers
- X allows apps to run on a network server but to be displayed on a desktop machine

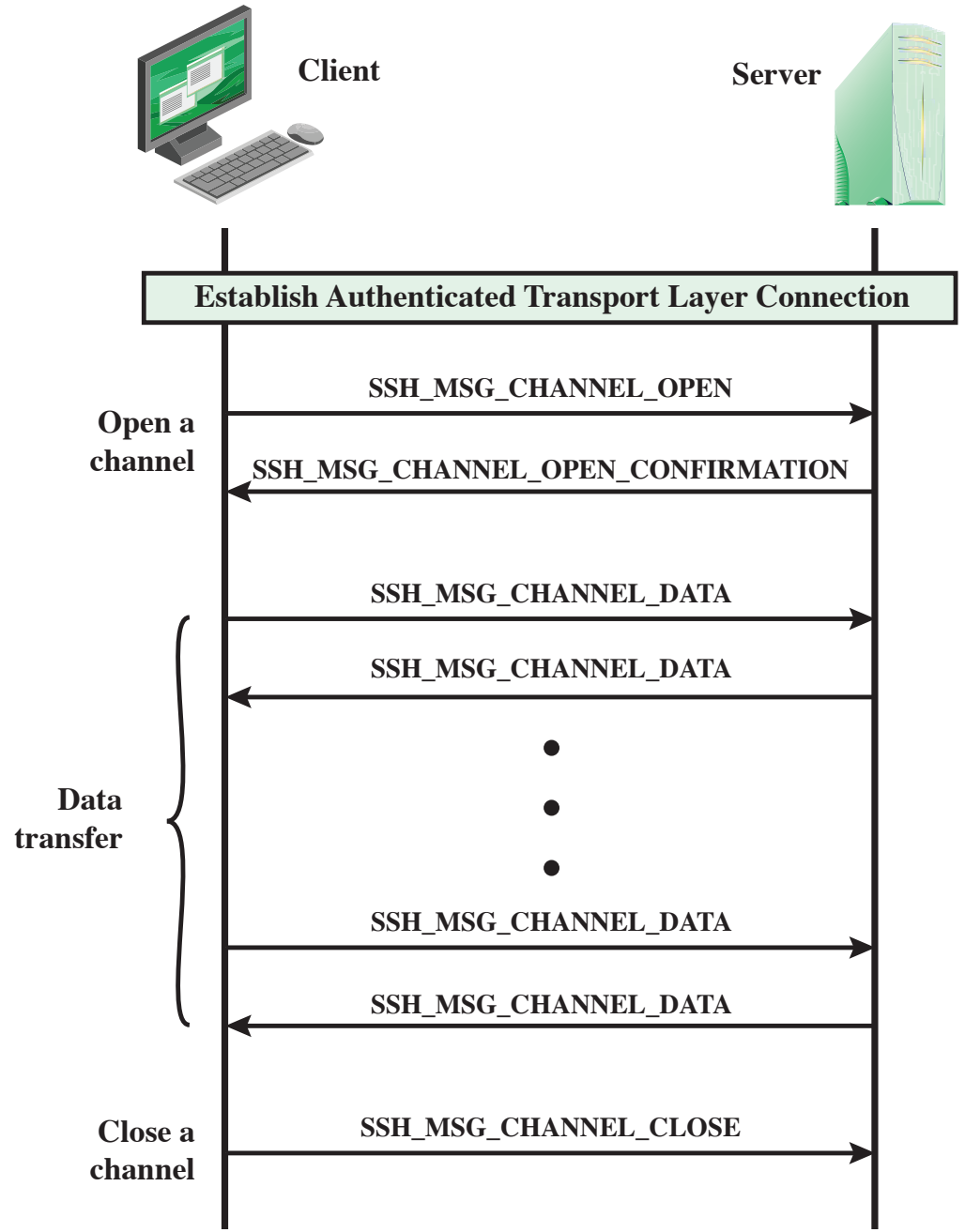
Forwarded-tcpip

- Remote port forwarding

Direct-tcpip

- Local port forwarding

SSH connection protocol exchanges



Connection protocol

- Opening a channel
  - SSH_MSG_CHANNEL_OPEN
    - Channel type
    - Sender channel number
    - Initial window size
    - Maximum packet size
    - Channel type specific data …

Connection protocol

- Opening a channel
  - SSH_MSG_CHANNEL_OPEN_CONFIRMATION
    - Recipient channel number (sender channel number from the open request)
    - Sender channel number
    - Initial window size
    - Maximum packet size
    - Channel type specific data …

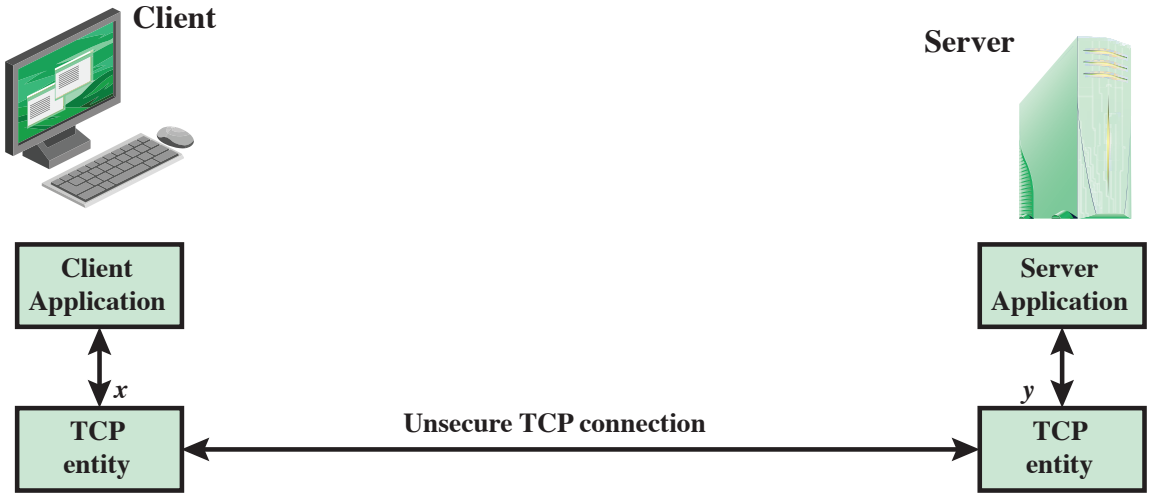
Connection protocol

- Data transfer over a channel
  - SSH_MSG_CHANNEL_DATA
    - Recipient channel number
    - Data
  - SSH_MSG_CHANNEL_CLOSE
    - Recipient channel

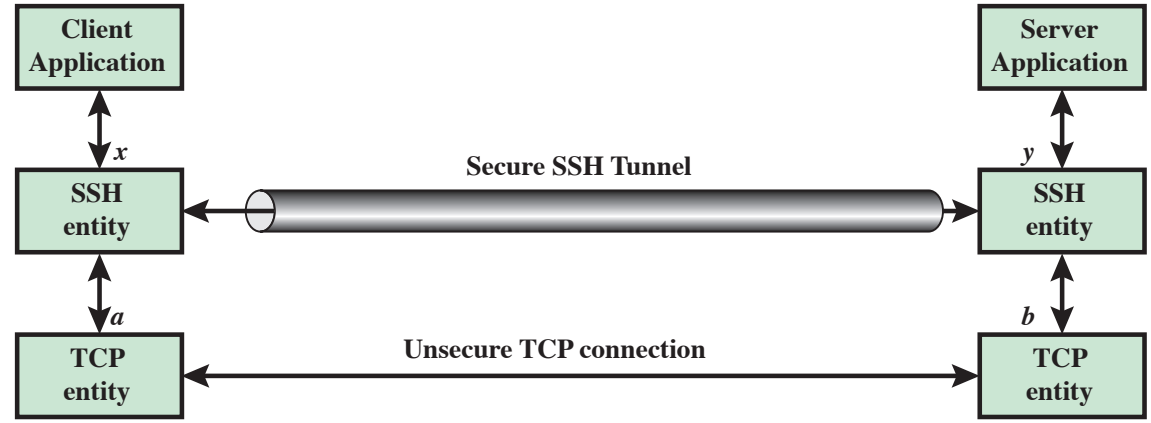
Port forwarding

- One of the most useful features of SSH
- Provides the ability to convert any insecure TCP connection into a secure SSH connection (also referred to as SSH tunneling)
- Incoming TCP traffic is delivered to the appropriate application on the basis of the port number (a port is an identifier of a user of TCP)
- An application may employ multiple port numbers

SSH TLP exchanges



(a) Connection via TCP



(b) Connection via SSH Tunnel

Προτεινόμενη βιβλιογραφία

- W. Stallings
  Cryptography and Network Security: Principles & Practice
  7th Ed., Prentice Hall, 2017