Chap. 6 Transport Layer Security

□ Web Security: SSL and TLS

□ HTTPS

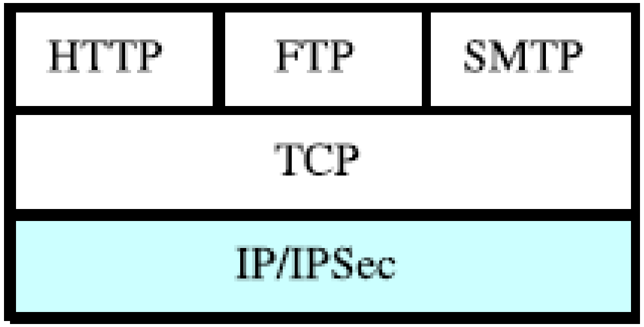
□ SSH

□ SET

Web Security Threats

- Integrity: modification of a web page, message traffic, or user data
- Confidentiality: eavesdropping of web traffic
- Denial of Service: bogus web requests, flooding web server memory or queue
- Authentication: impersonation of legitimate users

Security in TCP/IP Protocols

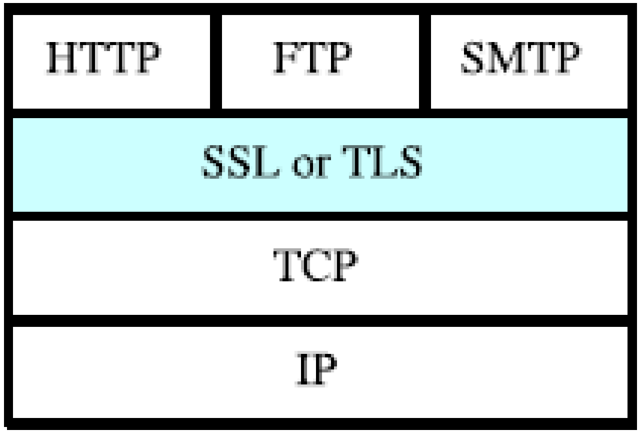


HTTP | FTP | SMTP

TCP

IP/IPSec

IP layer security



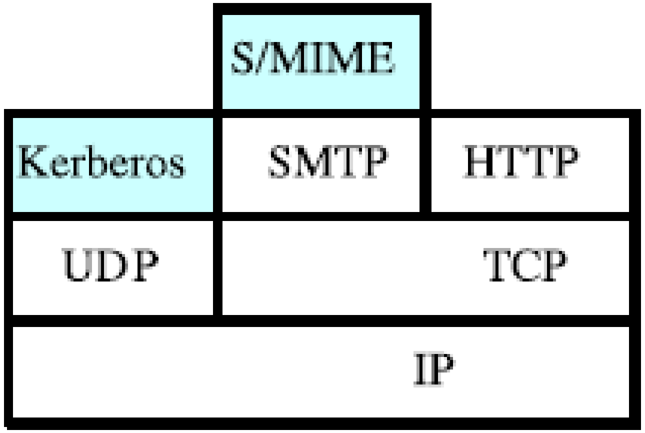
HTTP | FTP | SMTP

SSL or TLS

TCP

IP

Transport layer security



S/MIME

Kerberos | SMTP | HTTP

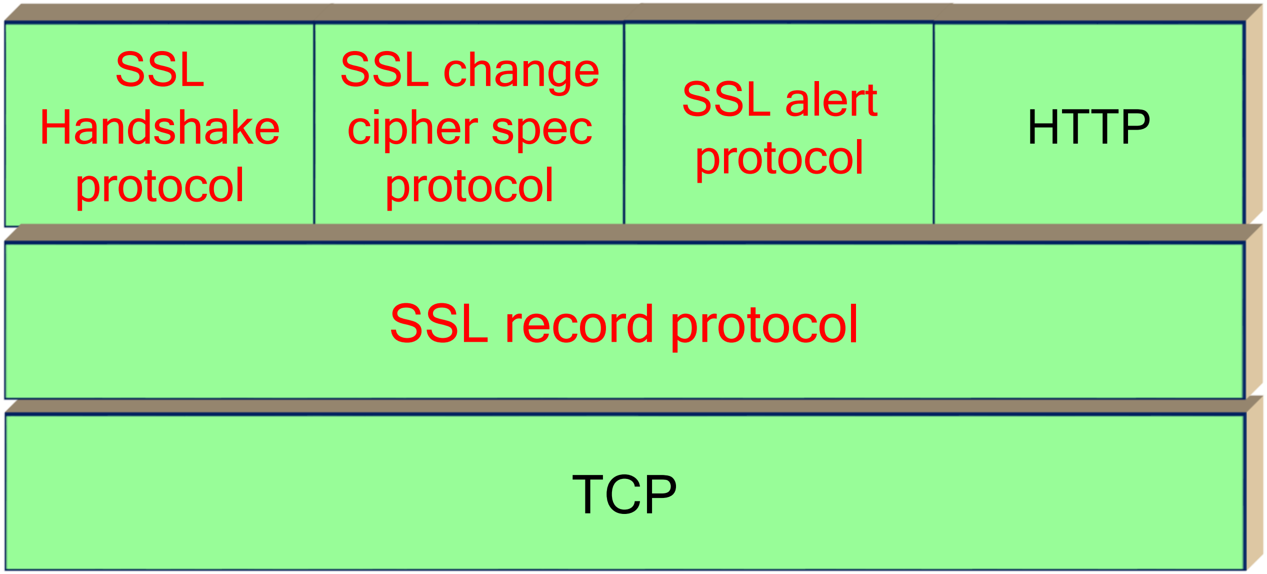
UDP | TCP

IP

Application layer security

Secure Socket Layer (SSL)

- Made by Netscape
- SSL architecture: provides a reliable end-to-end secure service based on TCP



SSL : Connection and Session

- □ Session
  - An association between a client and server for SSL transaction
  - Defined by two end-points and a set of cryptographic security parameters
  - created by a SSL handshake protocol
- □ Connection
  - transport connection: transient and peer-to-peer
  - Each connection is associated with a session
  - A session can consist of multiple connections

SSL : Connection and Session

- Session state:
  - Session ID: identify an active or resumable session
  - Peer certificate: X509.v3 certificate of the peer
  - Compression method
  - Cipher spec: defines encryption and hash algorithm
  - Master secret: 48-byte secret shared between client and server
  - Is_resumable: flag to indicate whether the session state is resumable

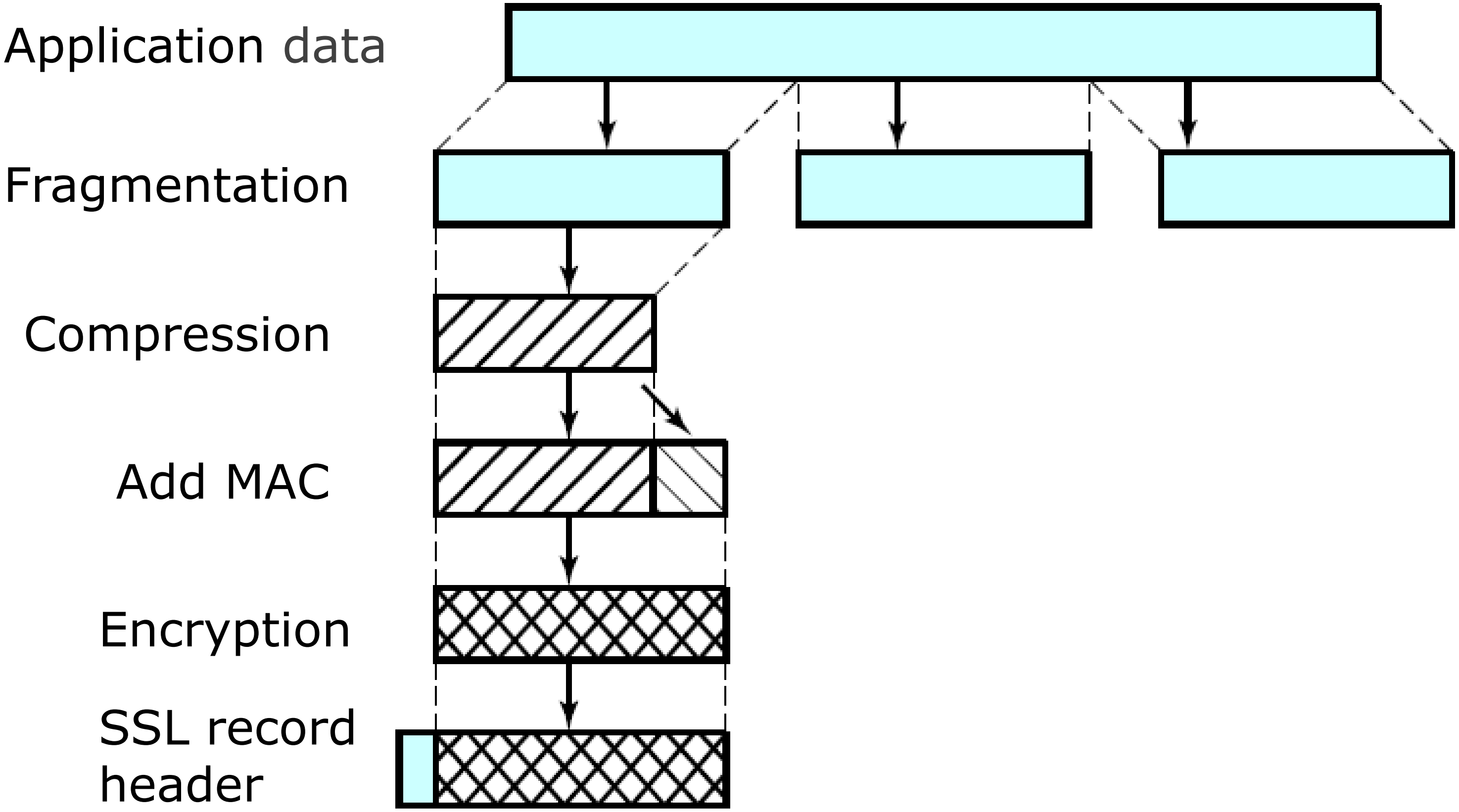
SSL : Connection and Session

□ Connection state:

- Server and client random: byte sequences chosen by the server and client for each connection
- Server (or client) write MAC secret: MAC secret key on data sent by the server (or client)
- Server (or client) write key: encryption key for data from server (or client) to client (or server)
- Initialization vector: IV for each key used in CBC DES
- Sequence numbers: sequence numbers for transmitted and received messages for each connection

SSL Record Protocol

- □ SSL record protocol provides confidentiality and message integrity
- □ SSL record protocol operation



Application data

Fragmentation

Compression

Add MAC

Encryption

SSL record header

SSL Record Protocol

- Message authentication
  - uses HMAC algorithm
  - MAC = H(write_MAC_secret || pad2 ||

    H(write_MAC_secret || pad1 || seq_num ||

    SSLCompressed.type || SSLCompressed.length ||
    SSLCompressed.fragment))

SSLCompressed.fragment: compressed fragment

SSLCompressed.type: compression type of the fragment

SSLCompressed.length: length of the fragment

SSL Record Protocol

Encryption

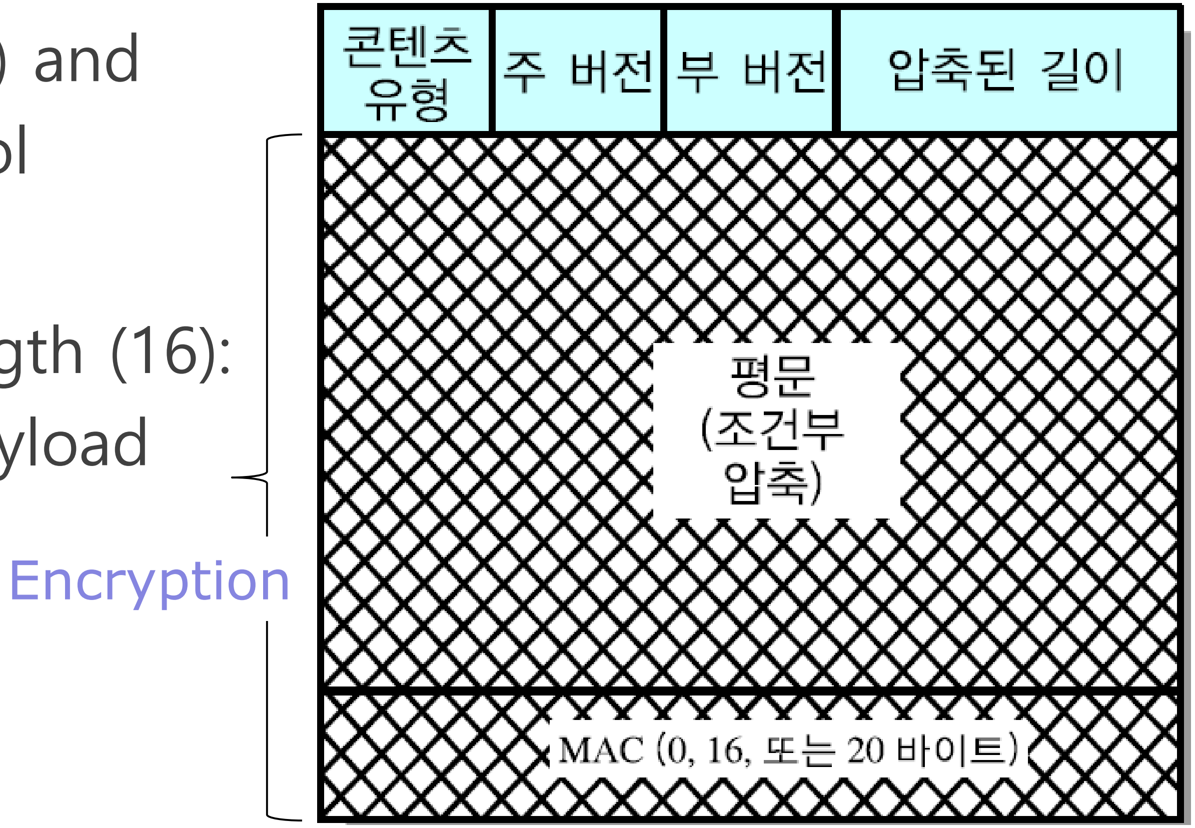
- Block cipher: AES, 3DES
- Stream cipher: RC4-128

- MAC : MD5 (SSL), HMAC (TLS)

SSL Record Protocol

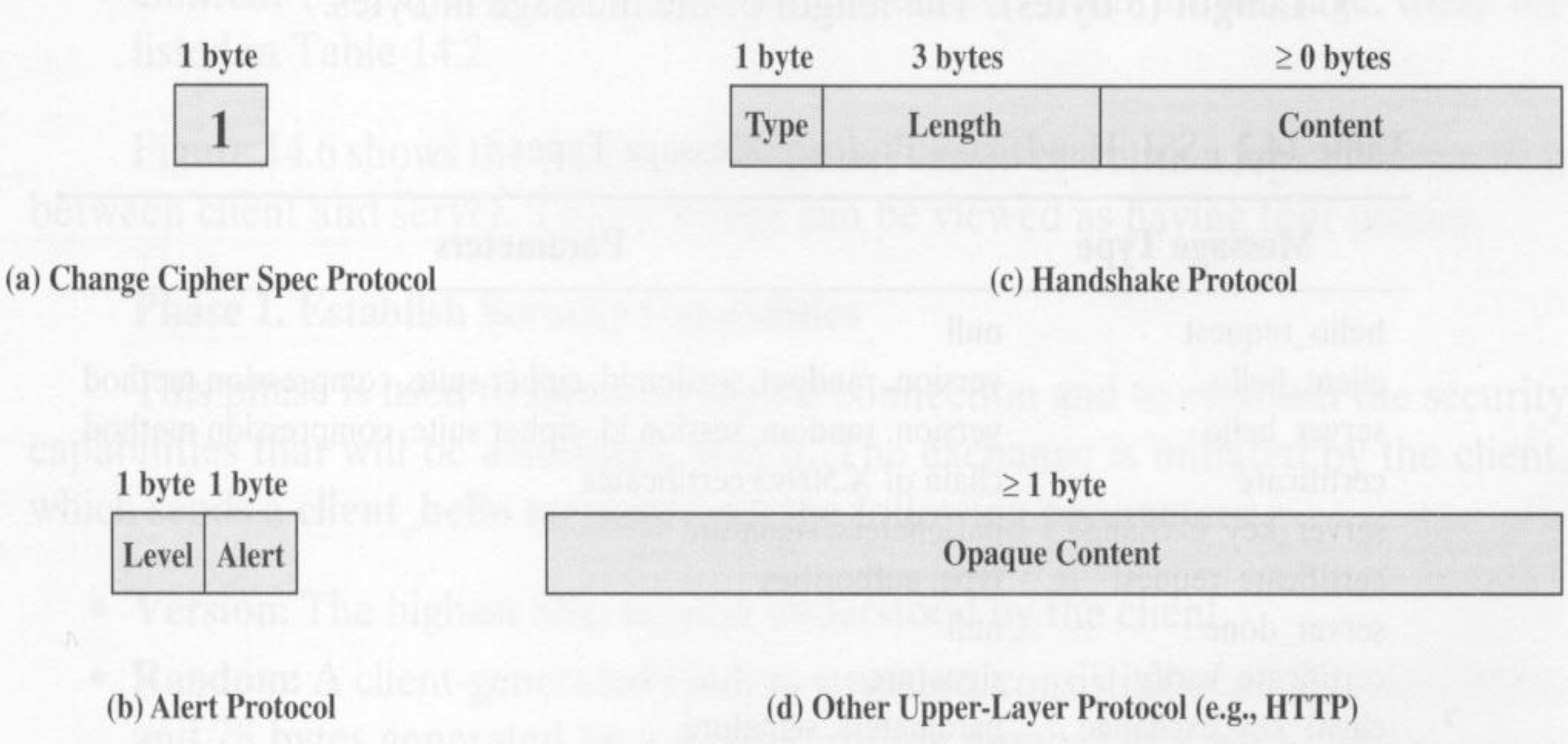
□ SSL record header

- Content type (8bits): higher layer protocol type; change_cipher_spec, alert, handshake, http
- Version: major(8) and minor(8) protocol version
- Compressed length (16): length of the payload



SSL Record Protocol

- □ SSL record protocol payload



(a) Change Cipher Spec Protocol

(c) Handshake Protocol

(b) Alert Protocol

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

SSL Record Protocol

□ SSL change cipher spec protocol

- change the state of the current SSL session from the pending state into the current state → activate the SSL session (SSL record protocol)

□ SSL alert protocol

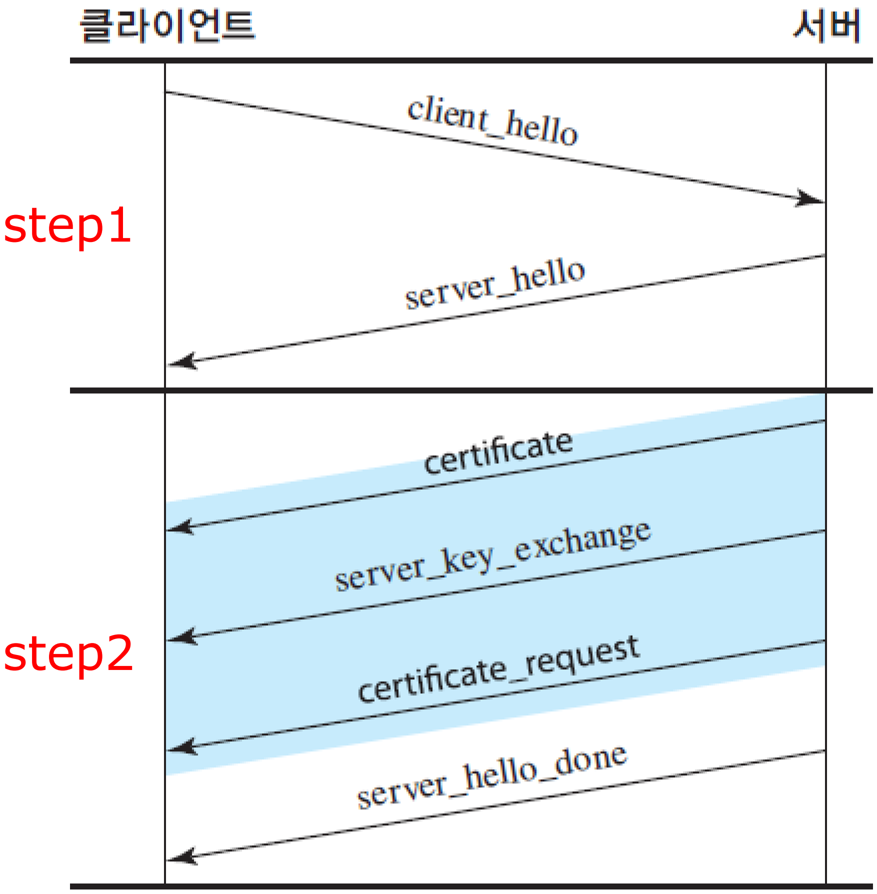
- Used to convey SSL-related alerts to the peer
- Example: unexpected_message, bad_record_mac, handshake_failure, decryption_failure, bad_certificate, certificate_revoked, certificate_expired, etc.

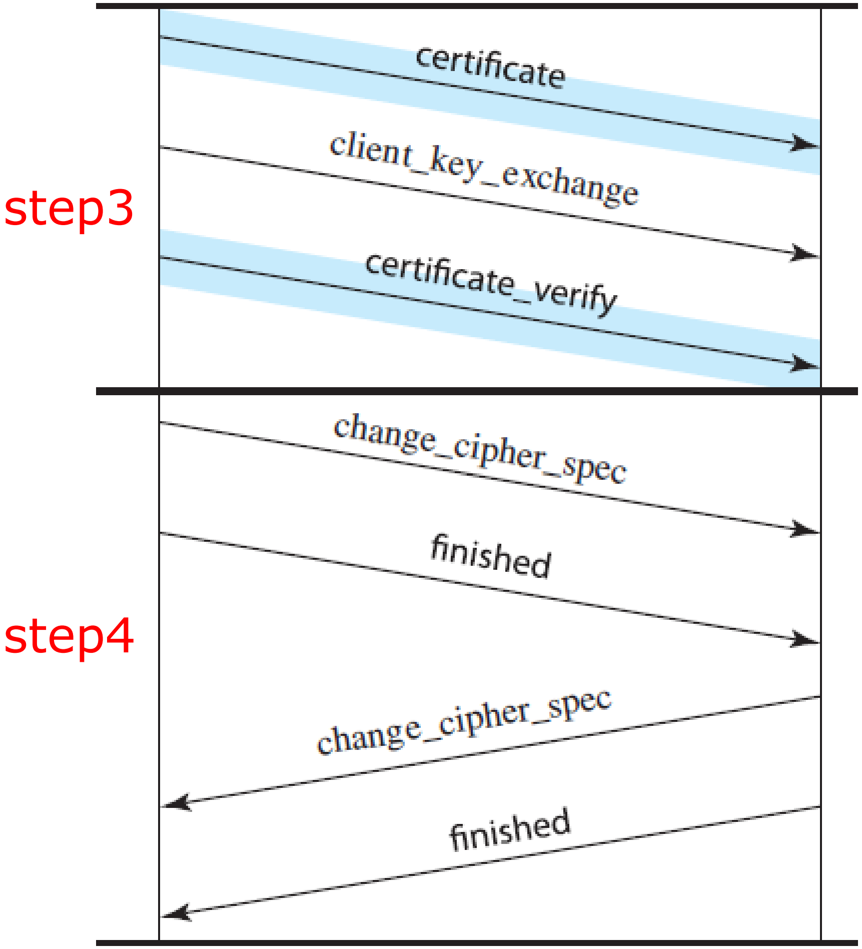
SSL Handshake Protocol

- □ Create an SSL session
  - ▪ Authenticate each other and negotiate cryptographic parameters such as encryption and MAC algorithms, cryptographic keys, etc.

- □ Step 1: establish security capabilities
- □ Step 2: server authentication and key exchange
- □ Step 3: client authentication and key exchange
- □ Step 4: finish

SSL handshake Protocol

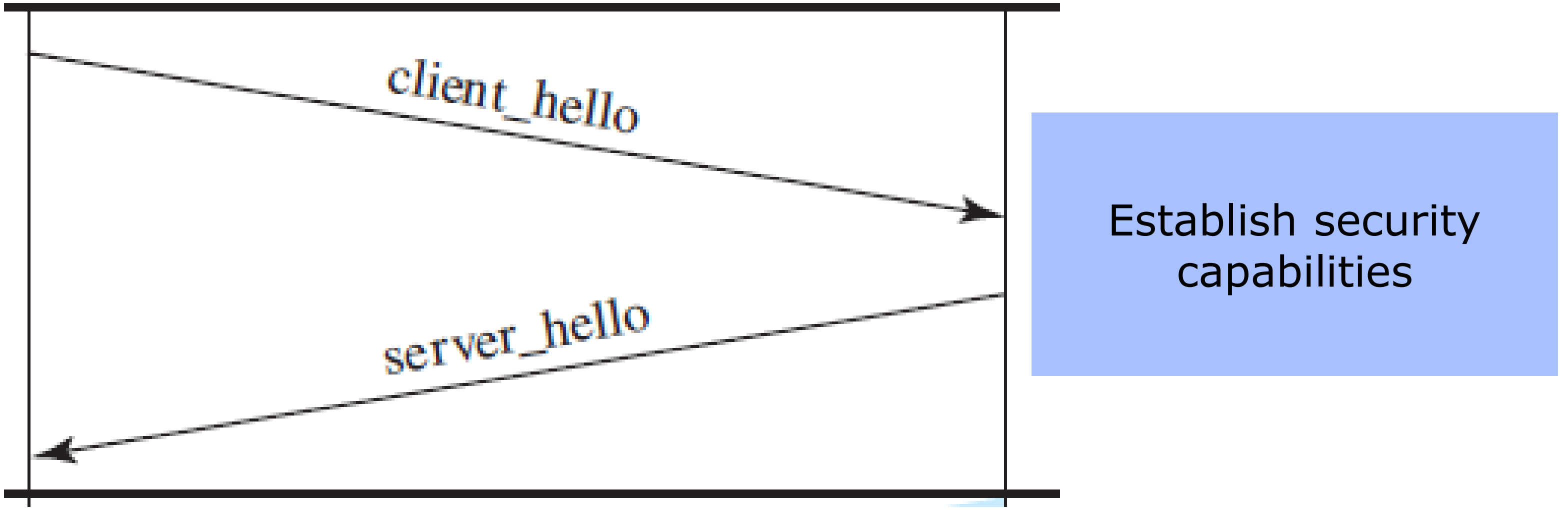




SSL Handshake Protocol

Phase 1: establish security capabilities

- □ Initiate a logical connection and establish the security capabilities
- □ client_hello -> server_hello



SSL Handshake Protocol

Phase 1: establish security capabilities

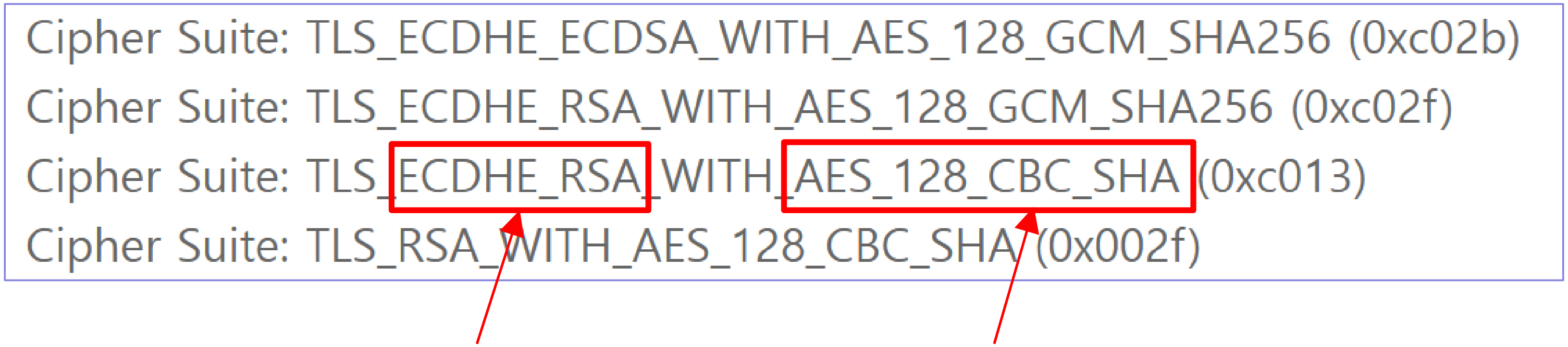
client_hello, server_hello message include:

- □ Random: nonces used in key generation to prevent replay attack
- □ SessionID: 0 (initiate a new connection on a new session), non zero(update the current security parameters); sessionID selected by the server
- □ CipherSuite: combinations of cryptographic algorithms supported by client; cipher spec chosen by the server
- □ CompressionMethod: list of compression methods supported by client; compression method chosen by server

SSL Handshake Protocol

Phase 1: establish security capabilities

- CipherSuite example



```
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
```

key-exchange
algorithm

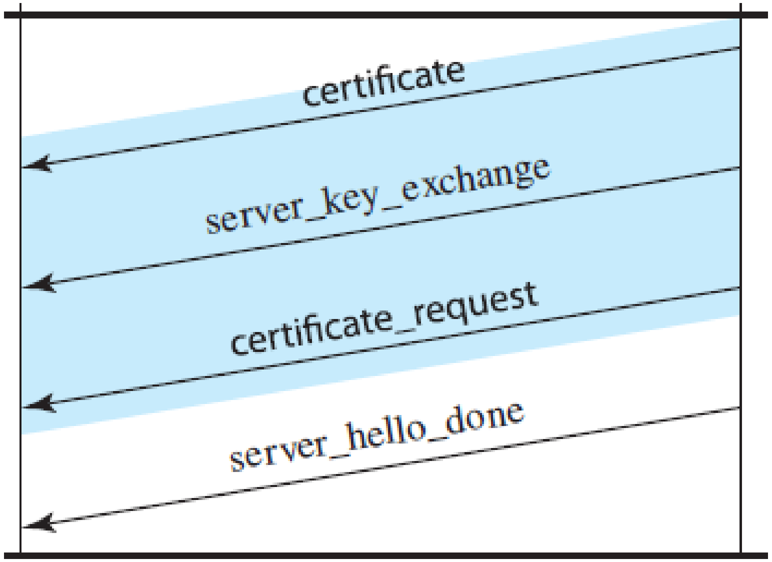
encryption and
hash algorithm

RSA: use RSA algorithm to send pre-master-secret
ECDHE_RSA: use ECDHE to send DH parameter with RSA-signed to generate pre-master-secret; provides forward secrecy

SSL Handshake Protocol

Phase 2: server authentication and key exchange

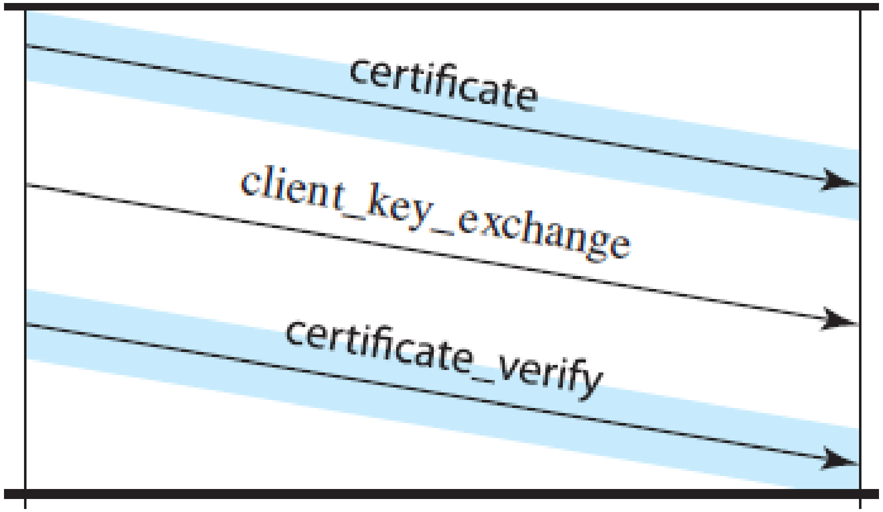
- □ certificate: sends server certificate
- □ server_key_exchange: sends server's key (Diffie-Hellmann public when DH key exchange is used)
- □ certificate_request: requests client's certificate
- □ server_done: indicates the end of the server hello and associated messages



SSL Handshake Protocol

Phase 3: client authentication and key exchange

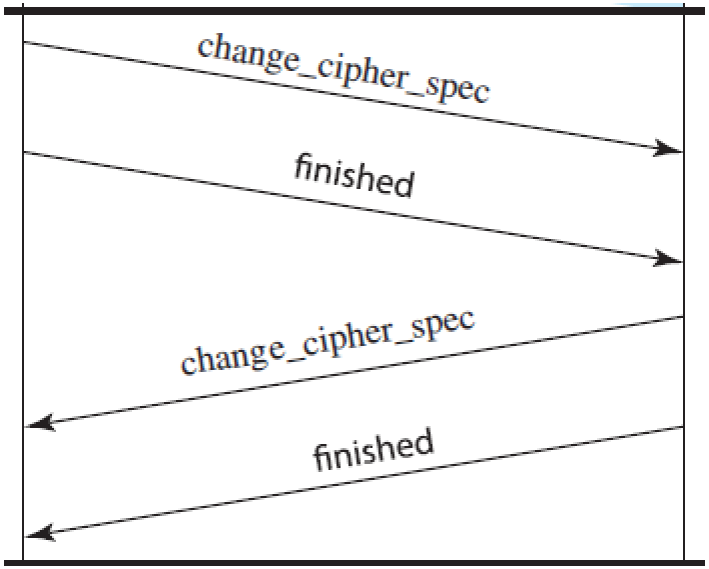
- □ certificate: sends client certificate when requested
- □ client_key_exchange: sends client's key (Diffie-Hellmann public when DH key exchange is used)
- □ certificate_verify: used to verify the client's certificate (signed hash for all handshaking messages exchanged from client_hello)



SSL Handshake Protocol

Phase 4: finish

- □ change_cipher_spec: copies the pending CipherSpec into the current CipherSpec
- □ finished: ends the handshake protocol



SSL Cryptographic Computations

- Generate Master secret
  - One-time 48-byte master secret for the current session by means of secure key exchange
  - Used to generate other cryptographic keys: client_write_MAC_secret, server_write_MAC_secret, client_write_key, server_write_key, and client_IV, server_IV

SSL Cryptographic Computations

- Master secret generation
  - Step 1 : generates and exchanges a pre_master_secret using RSA or Diffie-Hellmann
    - RSA: client generates a pre_master_secret, encrypts with server's public key, and sends to the server (client_key_exchange)
    - D-H: exchange D-H publics (server_key_exchange and client_key_exchange) and both sides calculate pre_master_secret independently

SSL Cryptographic Computations

- Master secret generation
  - Step 2 : client and server independently generates their master secrets using pre_master_secret and client and server randoms

SSL Cryptographic Computations

- Master_secret (48 Byte) =

  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))

SSL Cryptographic Computations

- Cryptographic keys
  - generate necessary keys from master_secret:
  - client_write_MAC_secret
  - server_write_MAC_secret
  - client_write_key
  - server_write_key
  - client_write_IV
  - server_write_IV

SSL Cryptographic Computations

□ key_block =

- MD5(master_secret || SHA('A' || master_secret || ServerHello.random || ClientHello.random)) || MD5(master_secret || SHA('BB' || master_secret || ServerHello.random || ClientHello.random)) || MD5(master_secret || SHA('CCC' || master_secret || ServerHello.random || ClientHello.random)) || MD5(master_secret || SHA(‘DDDD' || master_secret || ServerHello.random || ClientHello.random)) || . . .

HTTPS (HTTP over SSL)

- Combine HTTP and SSL for secure communication b/w web browser and server
- Use https:// and port 443
- Encryption
  - URL of the requested document
  - Contents of HTTP header, document and browser forms
  - Cookies between browser and server
- HTTP session – SSL Session – TCP conn.

HTTPS (HTTP over SSL)

- HTTP client: connection initiation
  - makes a TCP connection to https server (port 443)
  - performs SSL handshaking protocol to make a secure session (SSL session)
  - exchanges all HTTP protocol messages over the secure session

HTTPS (HTTP over SSL)

- HTTP session termination
  - sends HTTP request/response message including "Connection: close" header

  → SSL/TLS protocol: sends "close_notify" alert message to close SSL session

  → TCP protocol: connection termination

Homework

- SSL/TLS transaction message 캡쳐/분석
  - SSL/TLS 메시지 종류, 기능, 파라미터 등
  - wireshark 사용하여 https://server 접속 메시지 캡쳐 및 분석; 캡쳐된 메시지 분석
  - 메시지 포멧, 각 필드 의미 등

  - 리포트 파일 제출: “hw3-학번-이름.hwp”

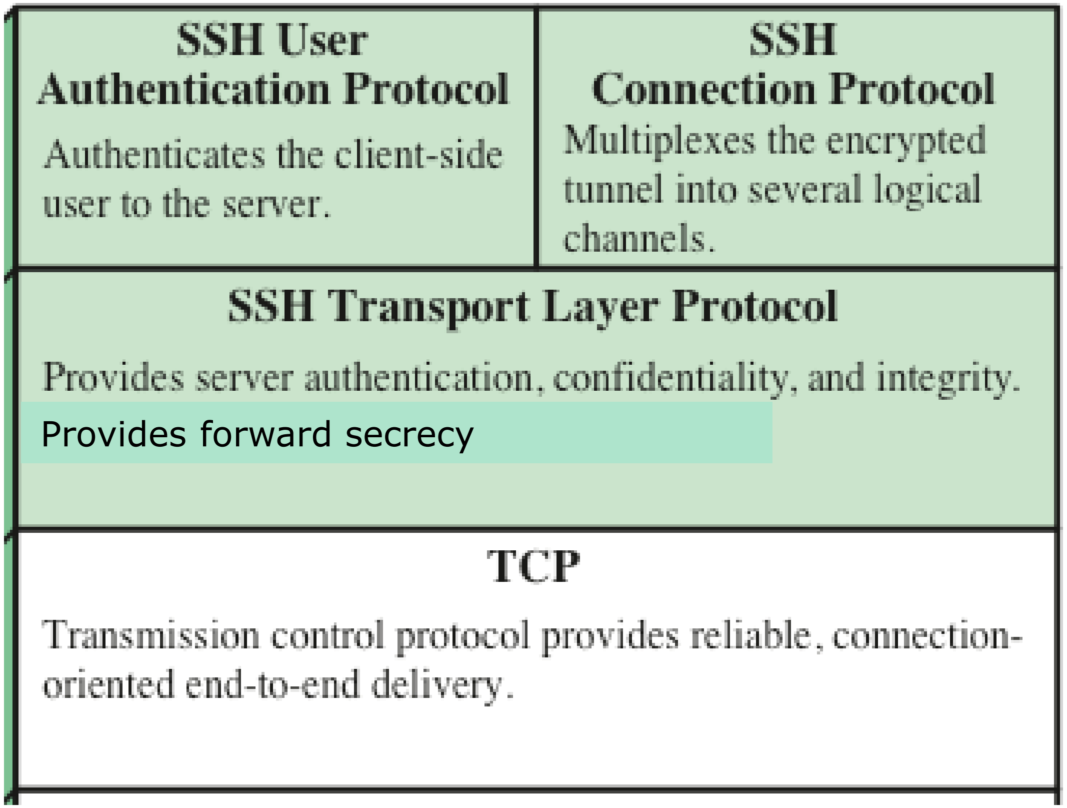
SSH (secure Shell)

- Protocol for securing remote communications like remote login, file transfer
  - SSH1 designed to provide a secure remote logon
  - SSH2 provides a more general secure client/server capability
  - RFC 4250-6

  - ssh –l pi raspberrypi
  - scp spidev.tar.gz pi@raspberrypi://home/pi/download

SSH (secure Shell)

- SSH protocols



SSH User Authentication Protocol

Authenticates the client-side user to the server.

SSH Connection Protocol

Multiplexes the encrypted tunnel into several logical channels.

SSH Transport Layer Protocol

Provides server authentication, confidentiality, and integrity.

Provides forward secrecy

TCP

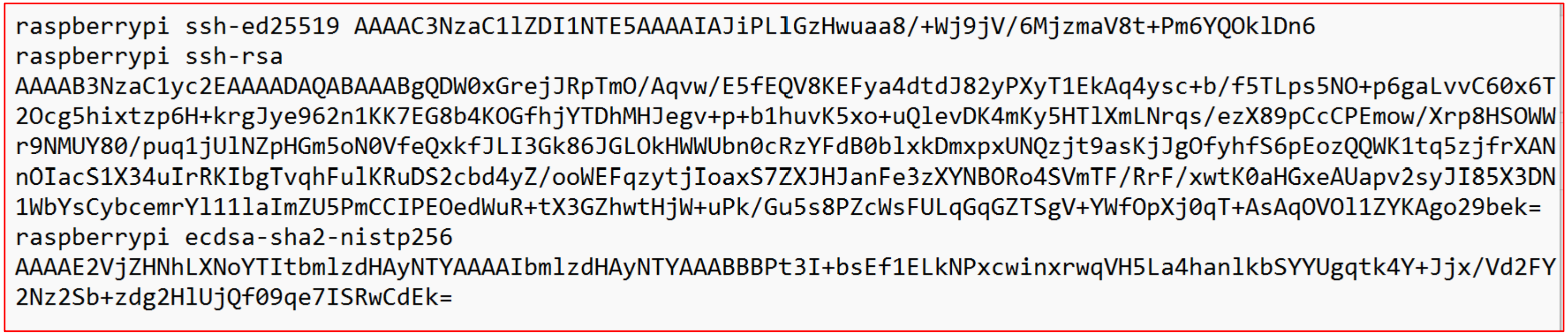
Transmission control protocol provides reliable, connection-oriented end-to-end delivery.

SSH (secure Shell)

- SSH protocols
  - SSH user authentication protocol: authenticates the client-side users to the server
  - SSH connection protocol: multiplexes multiple logical communications channels over a single SSH connection
  - SSH transport layer protocol: provides server authentication, data confidentiality, and data integrity with forward secrecy

SSH Transport Layer Protocol (TLP)

- Server authentication
  - server authentication based on server host keys (in /etc/ssh/*_key.pub, *_key pairs)
  - two public key trust models: RFC 4251
    - Client keeps a DB (host name, host public key): in “<user>/.ssh/known_hosts”
    - uses PK certificate: client has PK of CA



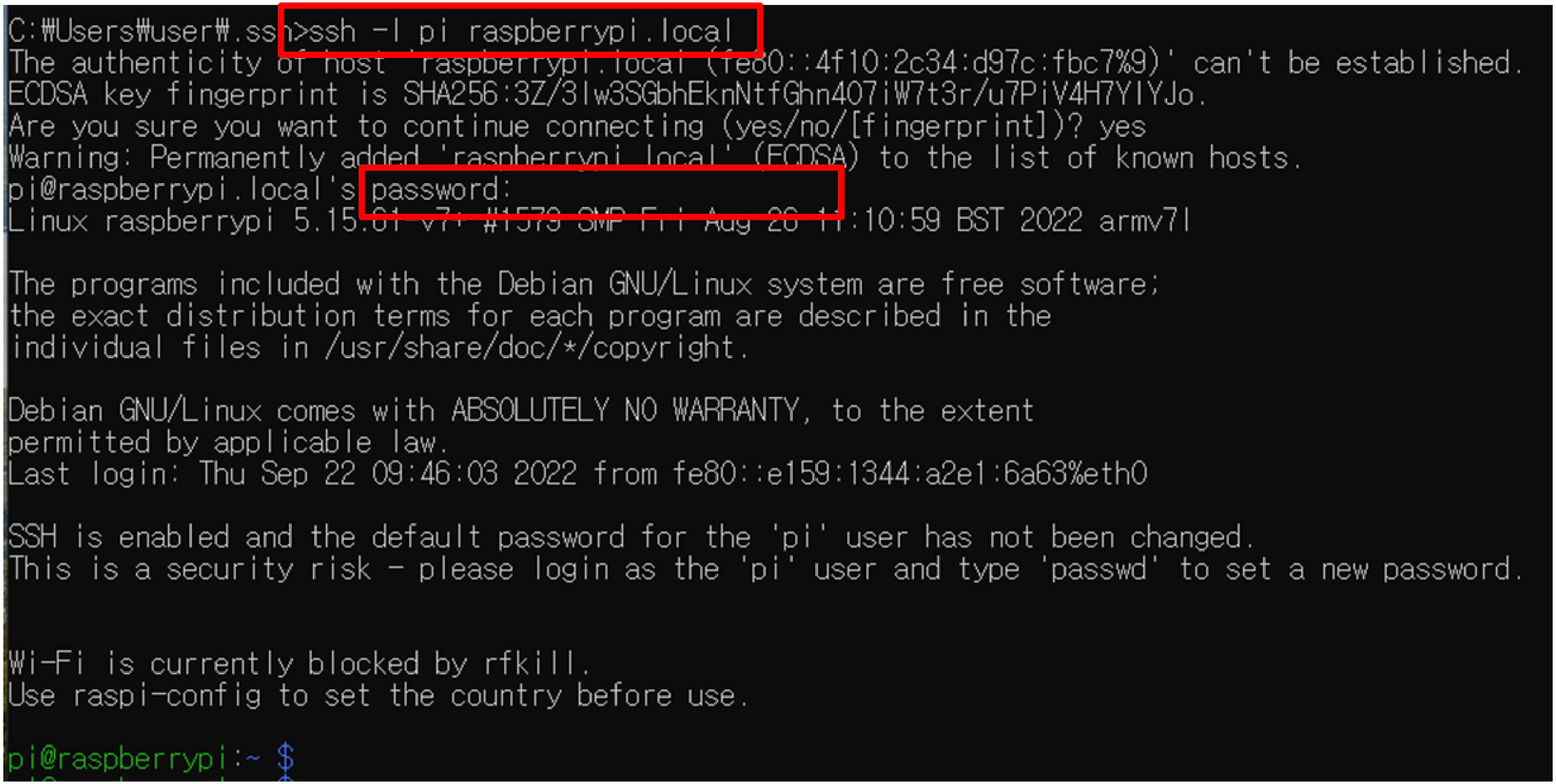
```
raspberrypi ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAJiPLlGzHwuaa8/+Wj9jV/6MjzmaV8t+Pm6YQOklDn6
raspberrypi ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDW0xGrejJRpTmO/Aqvw/E5fEQV8KEFya4dtdJ82yPXyT1EkAq4ysc+b/f5TLps5NO+p6gaLvvC60x6T
2Ocg5hixtzp6H+krgJye962n1KK7EG8b4KOGfhjYTDhMHJegv+p+b1huvK5xo+uQlevDK4mKy5HTlXmLNrqs/ezX89pCcCPEmow/Xrp8HSOWW
r9NMUY80/puq1jUlNZpHGm5oN0VfeQxkfJLI3Gk86JGLOkHWWUbn0cRzYFdB0blxkDmxpxUNQzjt9asKjJgOfyhfS6pEozQQWK1tq5zjfrXAN
nOIacS1X34uIrRKIbgTvqhFulKRuDS2cbd4yZ/ooWEFqzytjIoaxS7ZXJHJanFe3zXYNBORo4SVmTF/RrF/xwtK0aHGxeAUapv2syJI85X3DN
1WbYsCybcemrYl11laImZU5PmCCIPEOedWuR+tX3GZhwtHjW+uPk/Gu5s8PZcWsFULqGqGZTSgV+YWfOpXj0qT+AsAqOVOl1ZYKAgo29bek=
raspberrypi ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBPt3I+bsEf1ELkNPxcwinxrwqVH5La4hanlkbSYYUgqtk4Y+Jjx/Vd2FY
2Nz2Sb+zdg2HlUjQf09qe7ISRwCdEk=
```

SSH Transport Layer Protocol (TLP)

- Server authentication
  - server authentication process:
  - client: sends a nonce (N) after encrypting with server's public key
  - server: decrypts with it's private key and sends H(N)
  - client: computes H(N) and compares it with the received value

SSH Transport Layer Protocol (TLP)

- server authentication example



```
C:₩Users₩user₩.ssh>ssh -l pi raspberrypi.local
The authenticity of host 'raspberrypi.local (fe80::4f10:2c34:d97c:fbc7%9)' can't be established.
ECDSA key fingerprint is SHA256:3Z/3lw3SGbhEknNtfGhn407iW7t3r/u7PiV4H7YIYJo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'raspberrypi.local' (ECDSA) to the list of known hosts.
pi@raspberrypi.local's password:
Linux raspberrypi 5.15.61-v7+ #1579 SMP Fri Aug 26 11:10:59 BST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 22 09:46:03 2022 from fe80::e159:1344:a2e1:6a63%eth0

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.


Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~ $
```

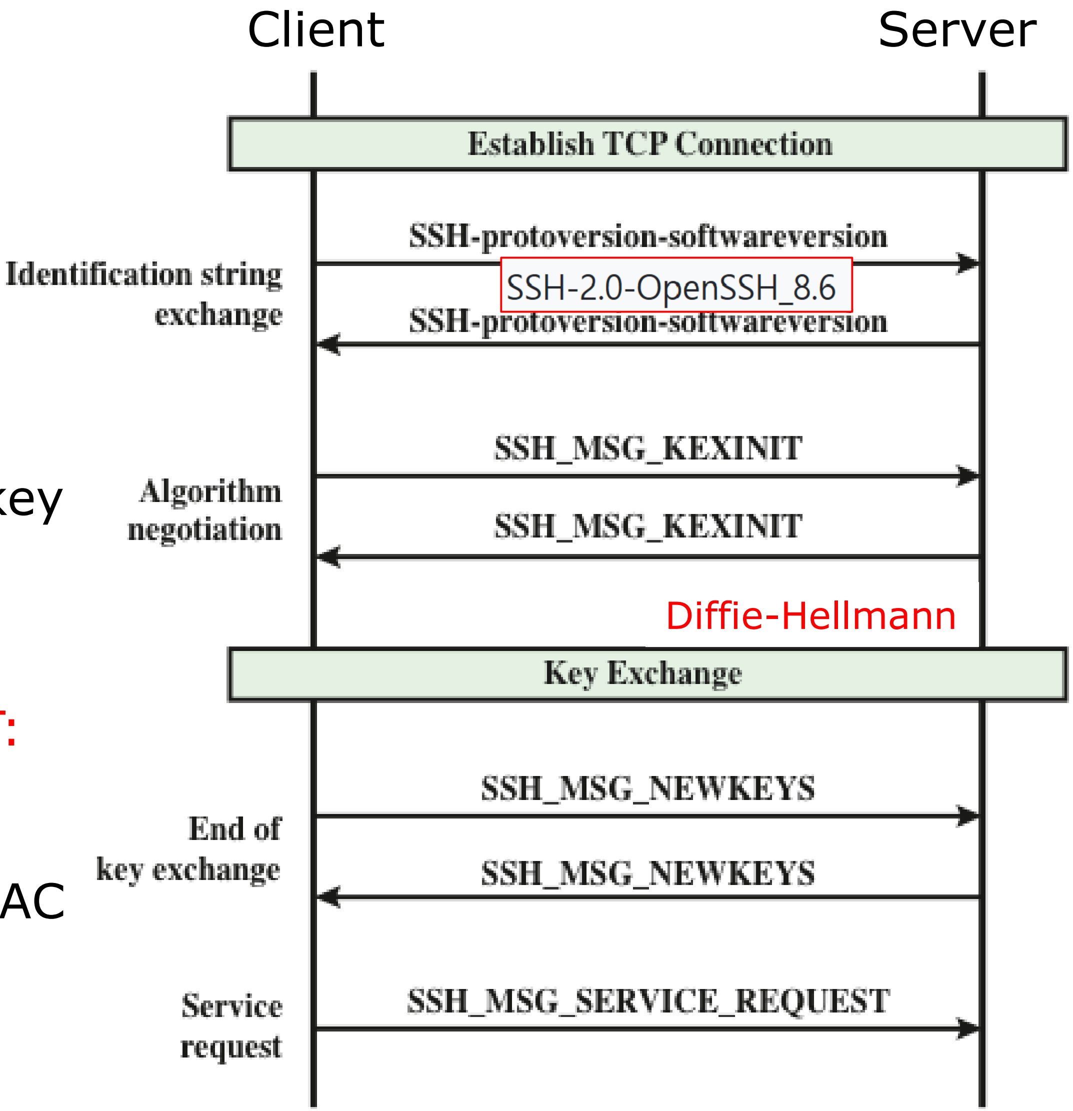
SSH Transport Layer Protocol (TLP)

- SSH TLP message exchange

Key Exchange: Generates a master key K for the session

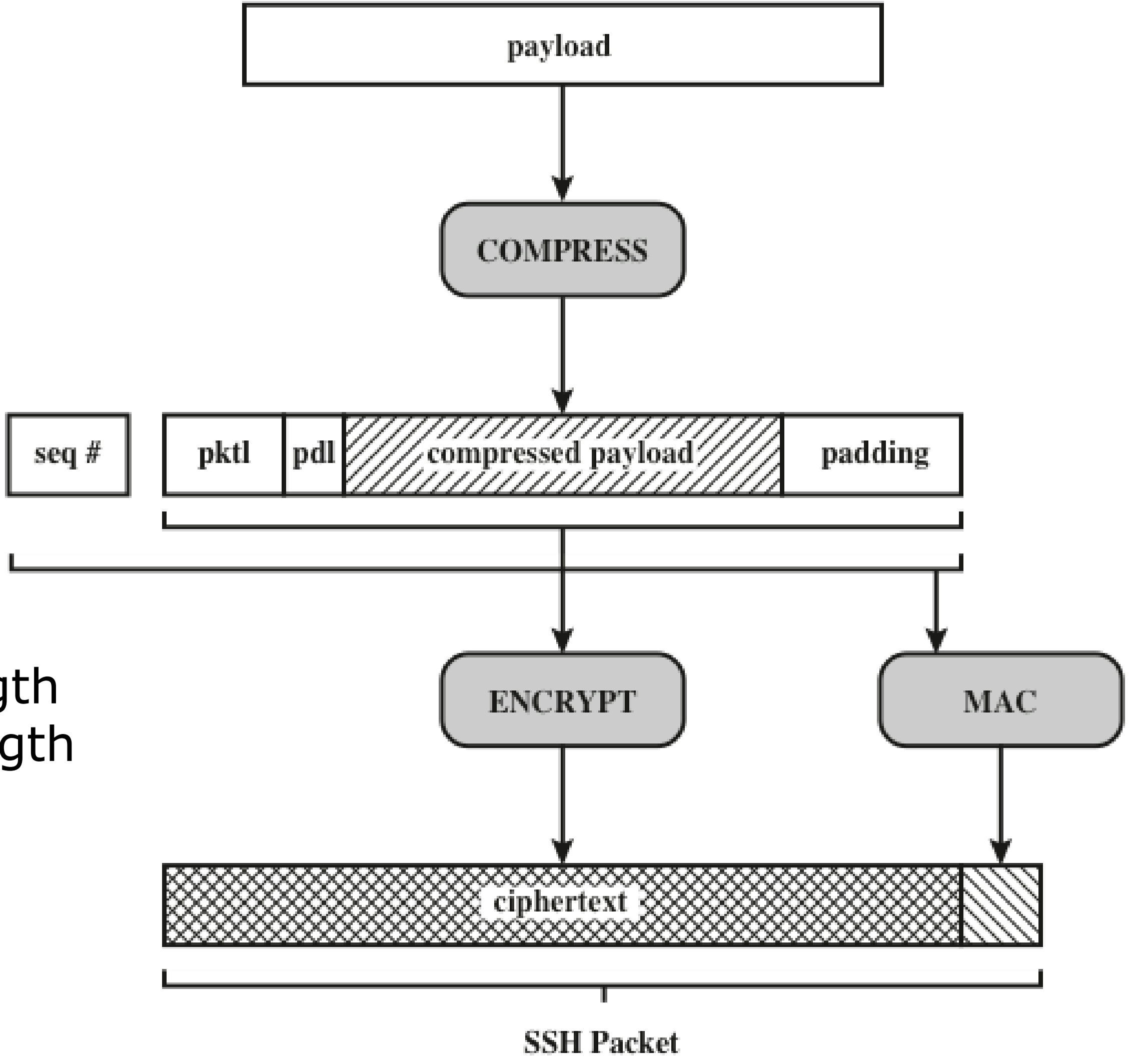
SSH_MSG_NEWKEYS: end of key exchange; from master key K, session keys are generated

SSH_MSG_SERVICE_REQUEST: sends User authentication or Connection protocol messages protected by encryption and MAC



SSH Transport Layer Protocol (TLP)

- SSH TLP packet



pktl: packet length
pdl: padding length

SSH Authentication Protocol

- SSH authentication protocol
  - used to authenticate the client to the server: password-based or public key-based authentication
  - password-based authentication:
    - sends a password with server's public key (in <user>/.ssh/known_hosts)
    - server decrypts the received value using it's private key and authenticates

SSH Authentication Protocol

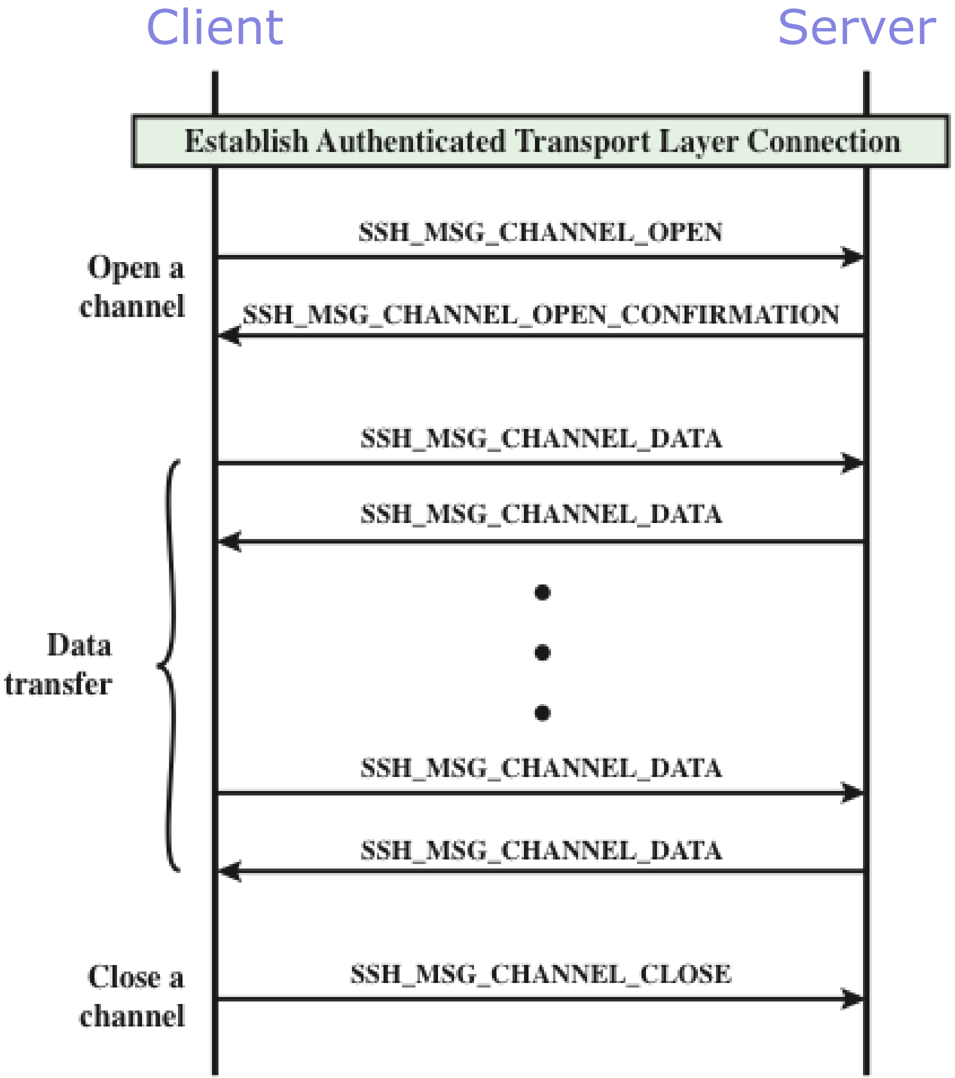
- SSH authentication protocol
  - public key-based authentication:
    - client's public-key has to be registered in it's home directory of ssh server (~/.ssh/authorized_keys)
    - client sends it's public-key and server finds a client's public key in home DIR
    - server generates a nounce and sends it after encrypting with client's public key
    - client decrypts it and sends it after encrypting with session key; server decrypts with session key

SSH Connection Protocol

- SSH connection protocol
  - Used to multiplex multiple logical channels (tunneling) on top of SSH TLP protocol after establishing a secure authentication connection
  - Channels are flow controlled using a window mechanism
  - Life of a channel: open channel – data transfer – close channel

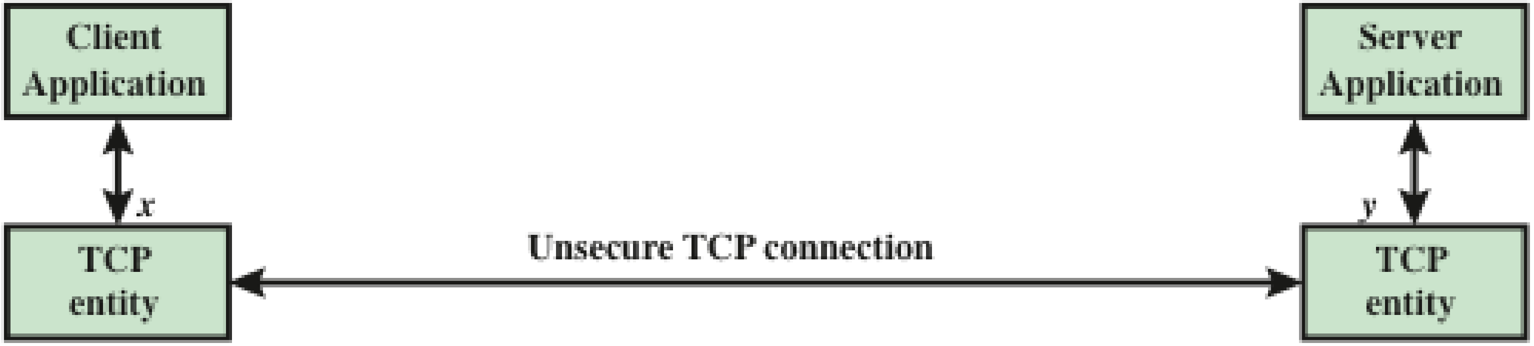
SSH Connection Protocol

- SSH connection protocol message exchange
- Channel type identifies an application
  - Session: remote execution of a program
  - X11: X window system
  - Port forwarding: local port forwarding, remote port forwarding



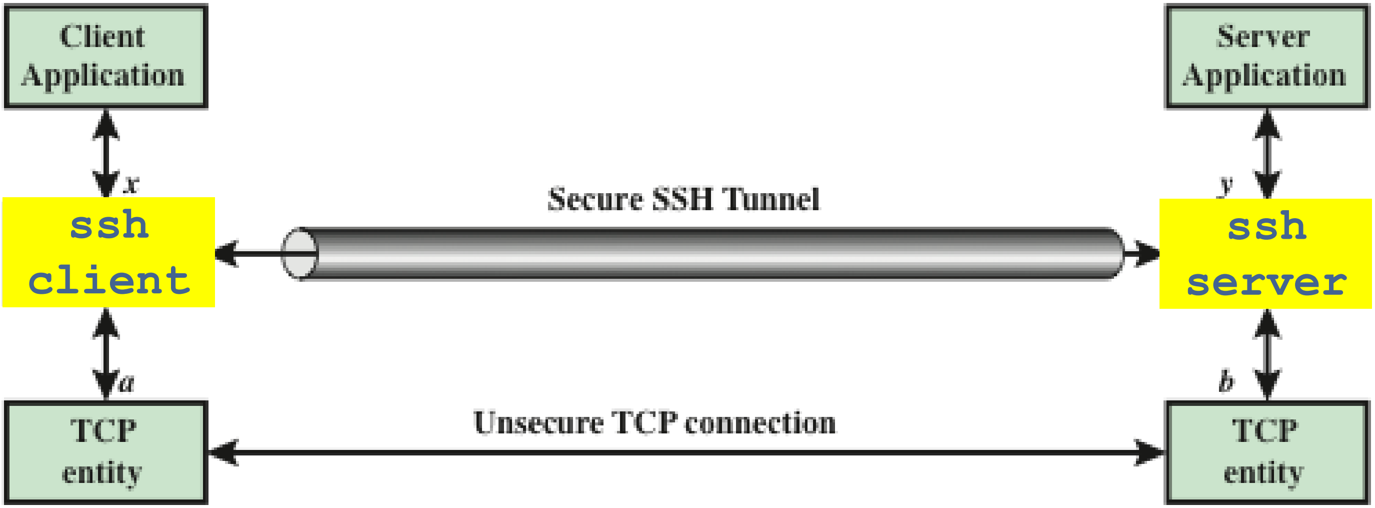
SSH Connection Protocol

- Unsecure channel using TCP
  - TCP connection between local port x and remote port y
  - provides no security



SSH Connection Protocol

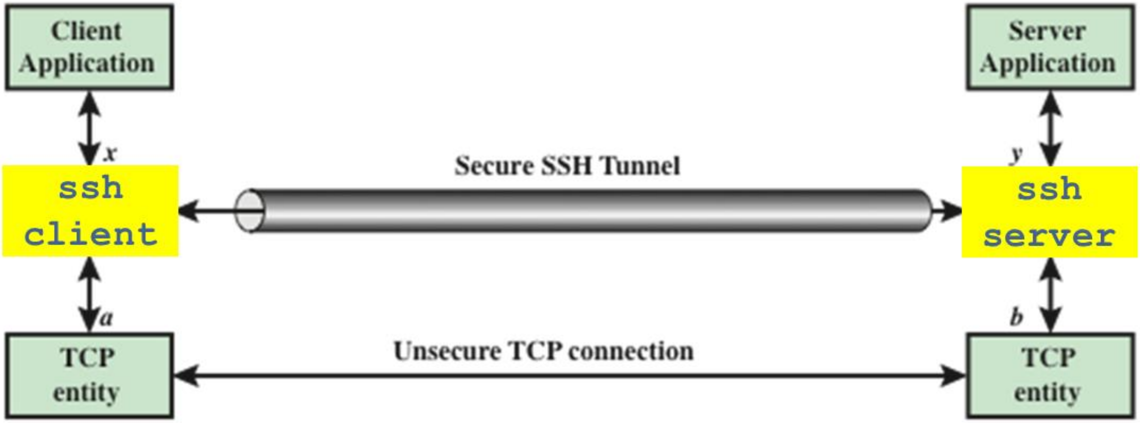
- Port forwarding (SSH tunneling)
  - Convert any insecure TCP connection into a secure SSH connection
  - SSH is configured to listen on the selected port and grabs all traffic using the port and sends it thru secure SSH channel



SSH Connection Protocol

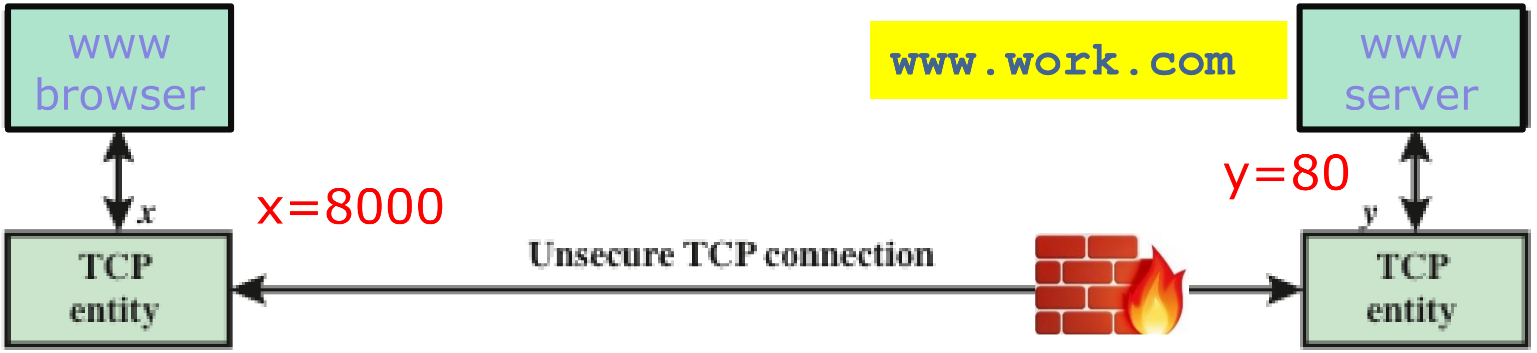
□ Local port forwarding

- connections from an SSH client are forwarded, via the SSH server, to a destination server
- SSH client: is configured to grab all traffic on the selected ports and sends it thru a secure SSH channel
- SSH server: sends the incoming traffic to the destination port



SSH Connection Protocol

- Local port forwarding example
  - web server access from outside thru ssh server
  - www browser side port 8000, www server port 80

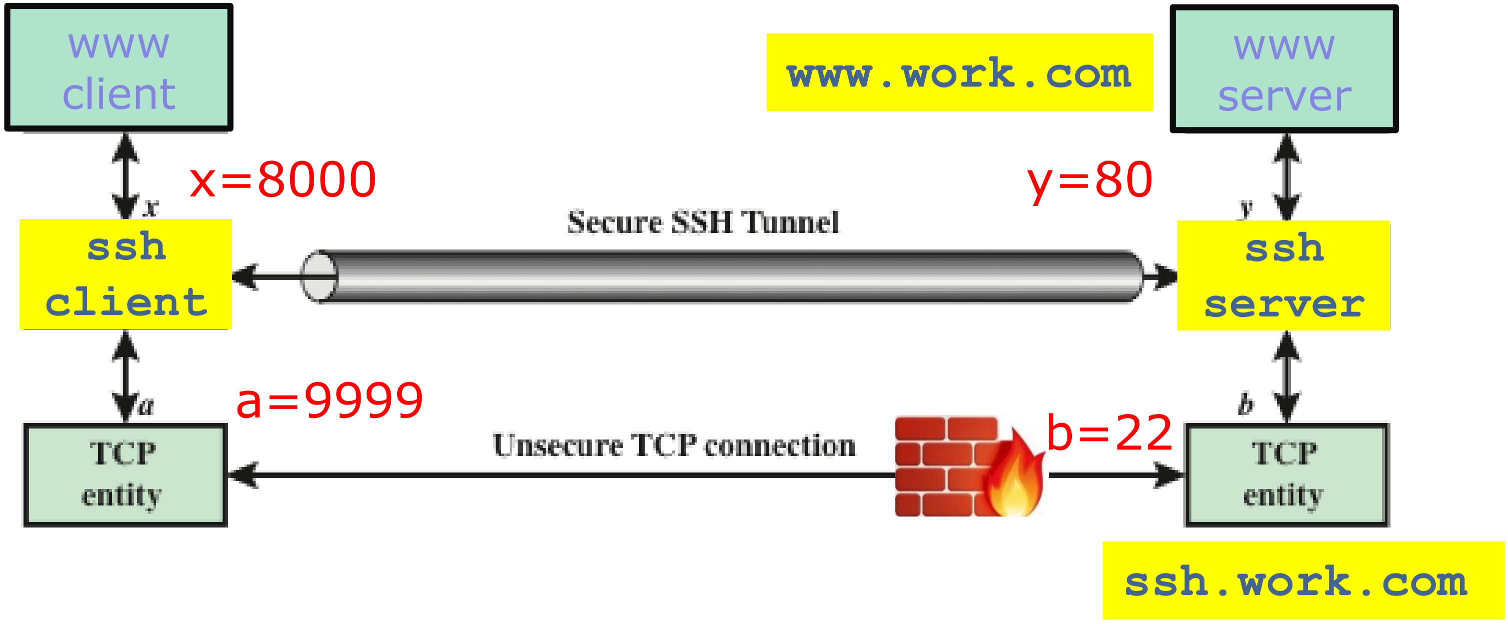


FW allows only ssh incoming connection (incoming connection to www server from outside is not allowed)

SSH Connection Protocol

- Local port forwarding example
  - www browser side port 8000, www server port 80
  - at ssh client side:

```
ssh -L 8000:www.work.com:80 mkk@ssh.work.com
```



SSH Connection Protocol

- Local port forwarding example
  - www browser side port 8000, www server port 80
  - at ssh client side:

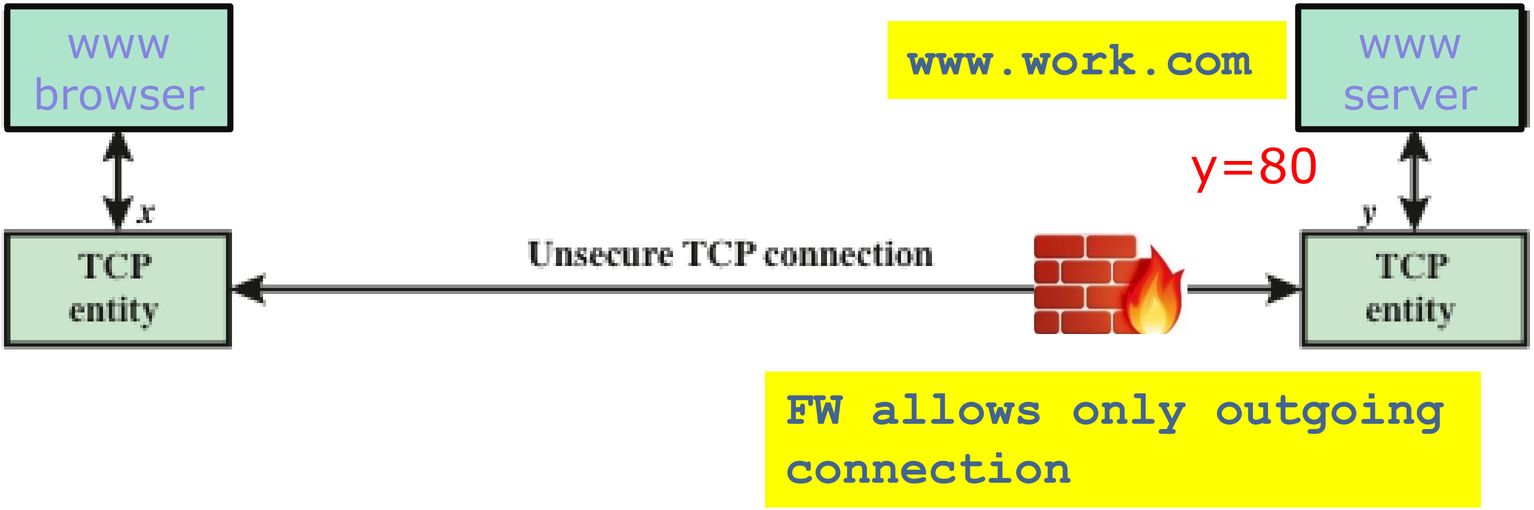
```
ssh -L 8000:www.work.com:80 mkk@ssh.work.com
```

1. SSH client sets up a SSH connection to remote SSH server over a TCP connection (port a=9999, port b=22)
2. SSH client is configured to accept traffic from 8000 to port 80 on the remote server (hijacking process) and sends it thru the ssh channel
3. SSH client informs SSH server to create a connection to the destination (to deliver the client application’s message to port 80 of www server)
4. SSH client takes traffic to local port 8000 and sends thru encrypted channel
5. SSH server receives incoming traffic, decrypts and sends to www server

SSH Connection Protocol

□ Remote port forwarding

- connections from an SSH server are forwarded, via the SSH client, to a destination server
- you want to access web server from the outside where firewall only allows outgoing connection

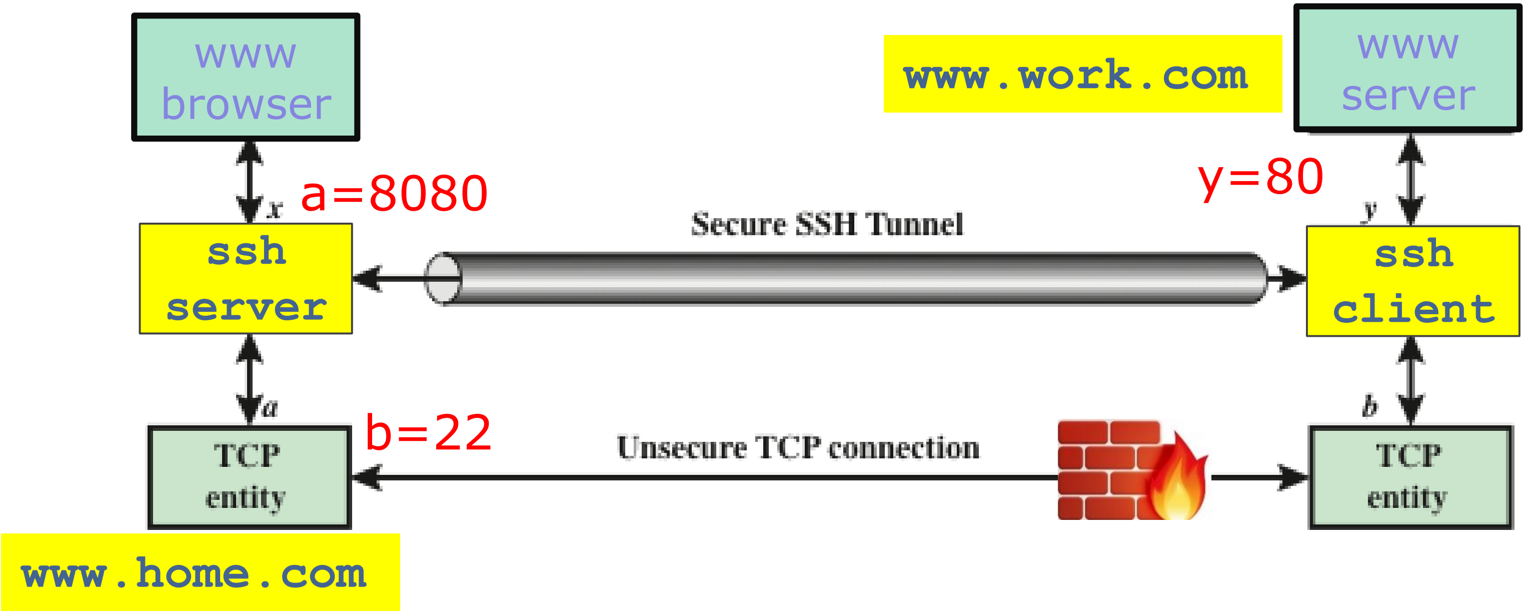


SSH Connection Protocol

□ Remote port forwarding

- firewall only allows outgoing connection
- you want to access web server from home computer
- on ssh client host:

```
ssh -R 8080:www.work.com:80 mkk@ssh.home.com
```



SSH Connection Protocol

□ Remote port forwarding example

```
ssh -R 8080:www.work.com:80 mkk@ssh.home.com
```

- SSH server is configured to accept all traffic destined to port 8080 and sends thru SSH tunnel
- SSH client accepts traffic from the SSH channel and delivers to port 80 on www.work.com

1. SSH client (work computer) sets up a SSH connection to remote SSH server (home computer) over a TCP connection (port a=9999, port b=22)
2. SSH server is configured to listen on local port (8080) to deliver traffic across the SSH connection
3. SSH client is configured to accept traffic from ssh channel and send www server on port 80
4. you access http://localhost:8080 on your browser of SSH server

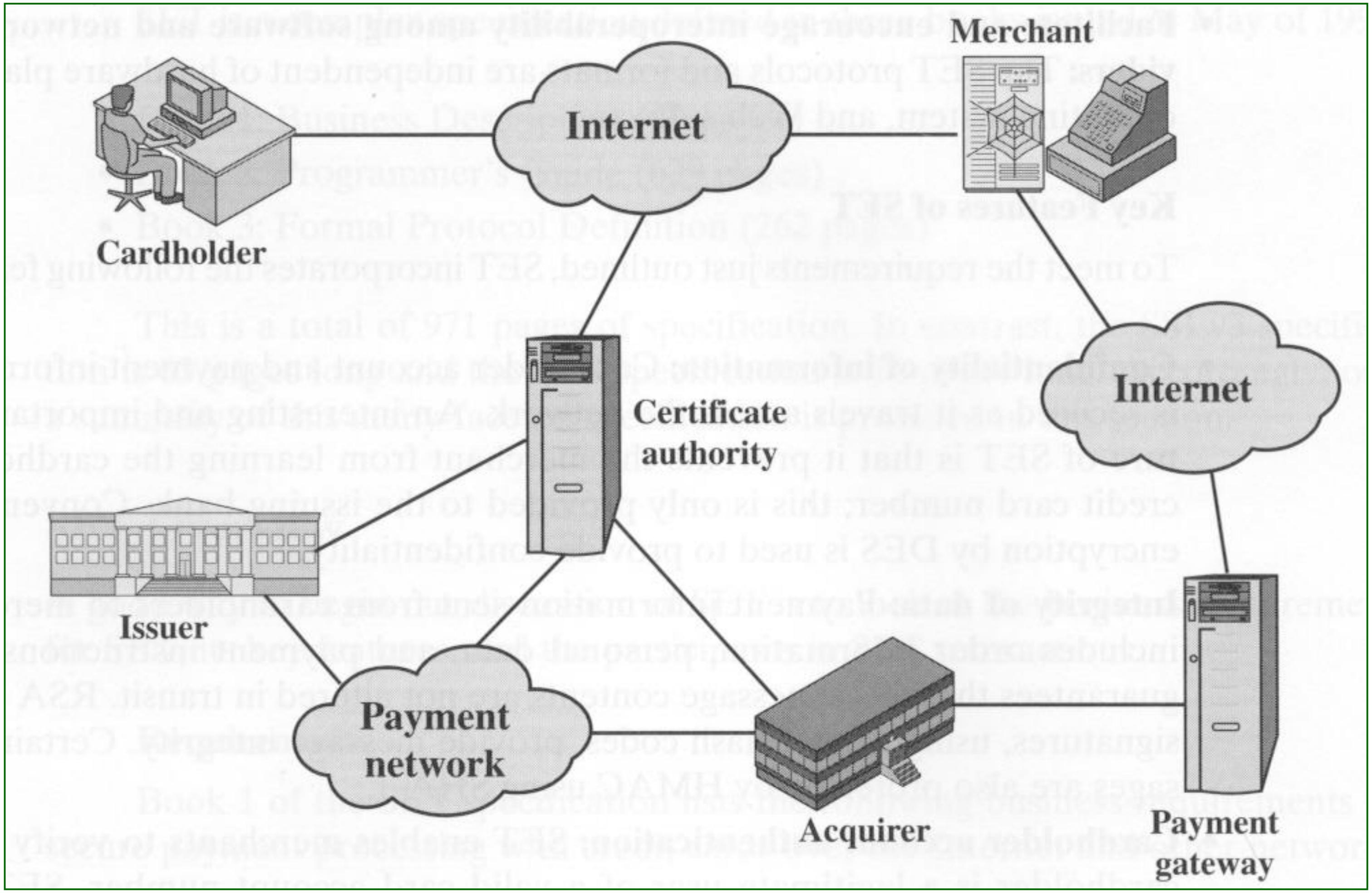
SET (Secure Electronic Transactions)

- security spec designed to protect credit card transactions over the Internet
- designed by Master and Visa card in 1996
- SET services
  - Secure communication among parties in a transaction
  - provides authentication, confidentiality, and integrity of communication
  - Trust model based on X.509v3 certificate
  - Ensure privacy

SET Features

- Confidentiality of information
  - DES encryption
  - Privacy:
    - prevents the merchant from learning cardholder’s credit card number
    - prevents the bank from learning cardholder’s order information
- Integrity of data
  - Uses digital signature: dual signature
- Authentication
  - Cardholder account and merchant authentication based on X.509v3 certificate with RSA signatures

SET Participants



SET Participants

- Cardholder
- Merchant
- Issuer:
  - a financial institute that provides the cardholder with the payment card
- Acquirer:
  - a financial institute that establishes an account with a merchant and processes payment card authorizations and payments

SET Participants

- Payment gateway:
  - a function that processes merchant payment messages;
  - interfaces b/w SET and the existing bankcard payment networks
- Certification authority:
  - an entity that issues X.509v3 certificate to cardholders, merchant, and payment gateways

SET Transactions

1. The customer opens an account and receives a certificate
2. The merchant has his certificates

3. Customer requests an order form
4. The merchant sends an order form and its certificate -> customer verifies the merchant certificate

5. Customer sends the order and payment information along with his certificate

SET Transactions

6. Merchant requests payment authorization to the payment gateway

7. Payment gateway confirms the payment to the Merchant

8. Merchant confirms the order to the customer and sends the goods or service

9. Merchant requests payment to the payment gateway

Dual Signature (DS)

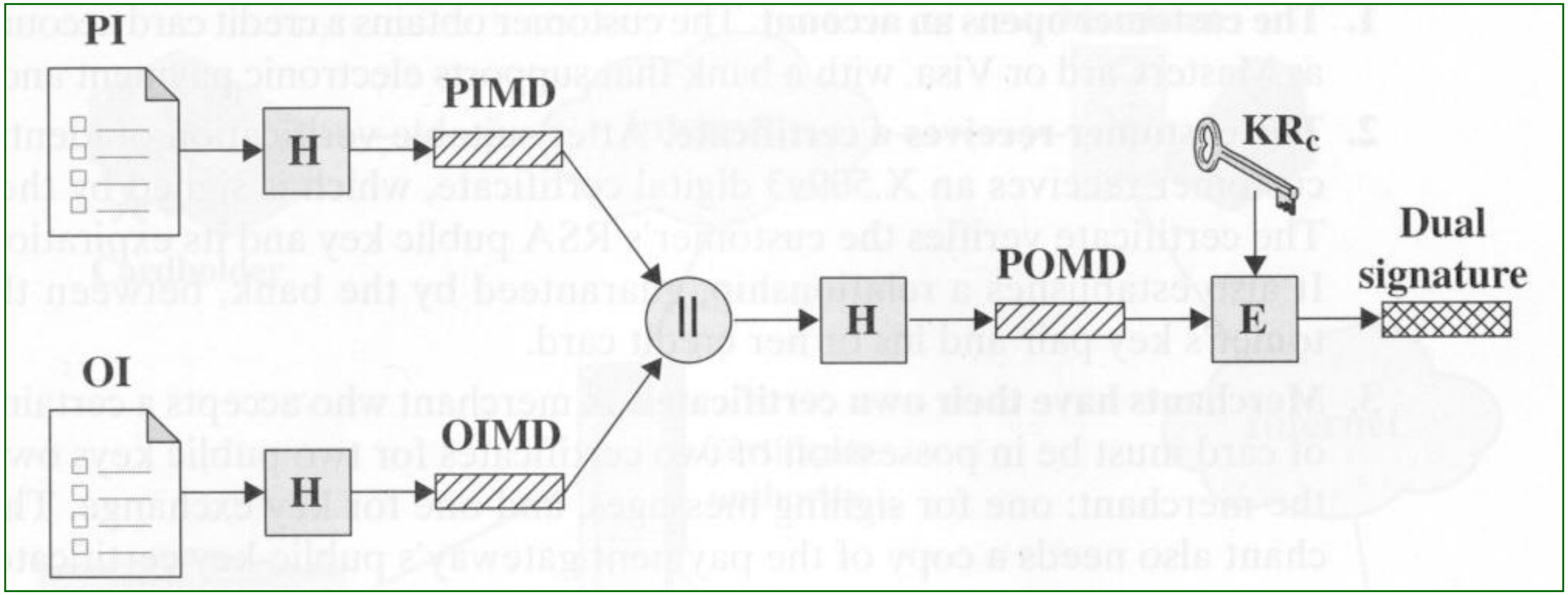
- The customer needs to send order info (OI) and payment info (PI)
- But, PI (credit card number and secret code) must be concealed from the merchant and
- OI (product information) must be concealed from the bank

- But, the two OI and PI must be linked

Dual Signature

- Dual signature:

$$DS = E_{KR_c}[H(H(PI) \| H(OI))]$$



Dual Signature

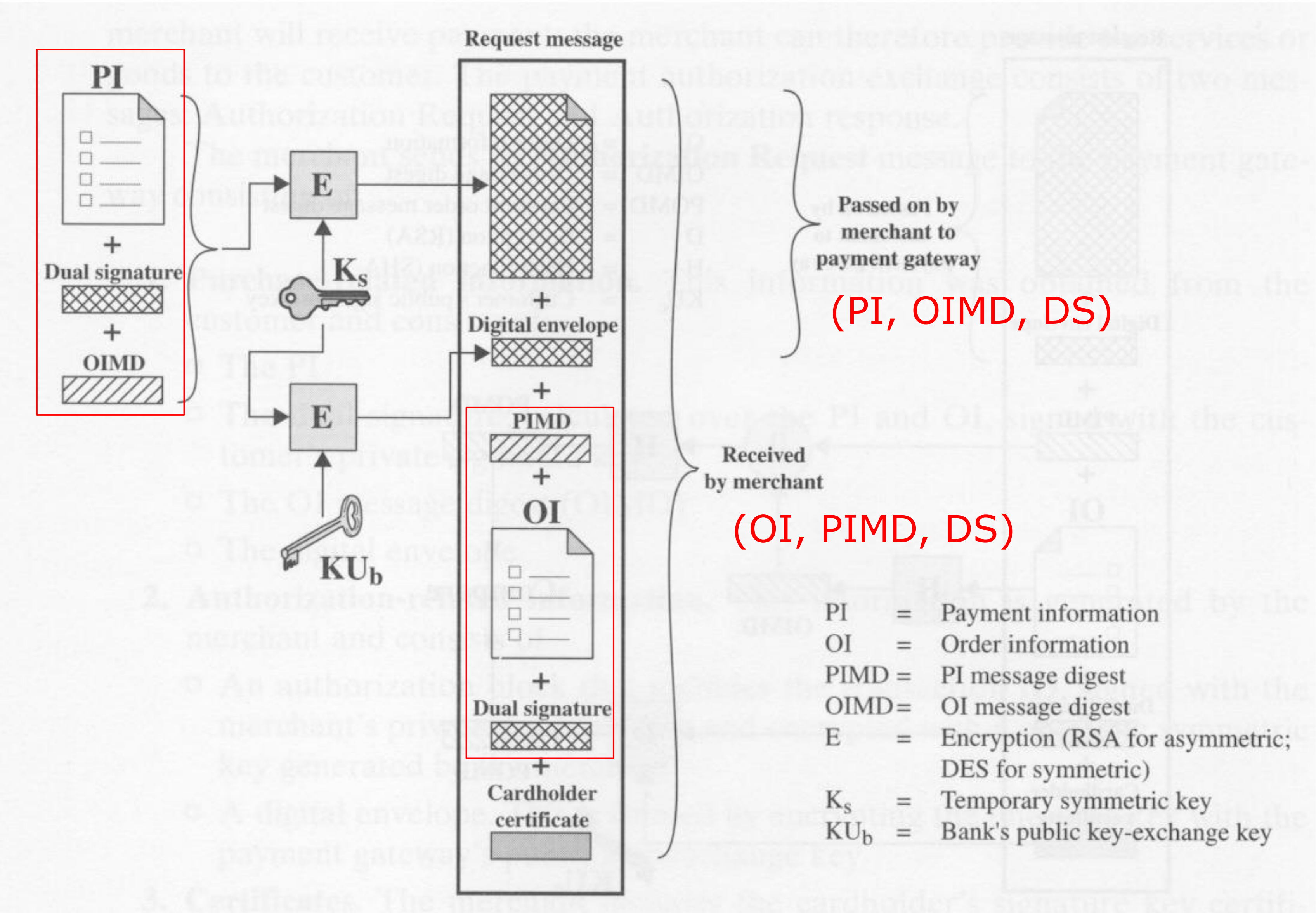
- □ Merchant is given DS, OI, and PIMD
  - Merchant computes H[PIMD || H(OI)] and $D_{KU_c}[DS]$
  - The signature is verified if the two are equal
  - The merchant cannot see the details of PI

- □ Bank is given DS, PI, and OIMD
  - Bank computes H[H(PI) || OIMD] and $D_{KU_c}[DS]$
  - The signature is verified if the two are equal
  - The bank cannot see the details of OI

Payment Processing

- Purchase request
  - Initiate request: request certificate to the merchant
  - Initiate response: merchant sends signed response, his certificate, and the payment gateway's certificate
  - Purchase request: cardholder sends a purchase request including purchase-related info (PI, OIMD, DS), order-related info (OI, PIMD, DS), and his certificate
  - Purchase response: signed response from the merchant for the purchase request

Purchase request message



(PI, OIMD, DS)

(OI, PIMD, DS)

PI	=	Payment information
OI	=	Order information
PIMD	=	PI message digest
OIMD	=	OI message digest
E	=	Encryption (RSA for asymmetric; DES for symmetric)
K_s	=	Temporary symmetric key
KU_b	=	Bank's public key-exchange key

Payment Authorization

- Payment authorization
  - ensures that the merchant will receive the payment
  - For payment authorization, Merchant sends authorization request message to payment gateway
- Authorization request message includes
  - Purchase-related info: (PI, OIMD, DS) and digital envelope
  - Authorization-related info: (transaction ID signed with merchant’s private key) and digital envelope
  - Certificates: cardholder’s signature key certificate, merchant’s signature key certificate

Payment Authorization

- Payment gateway authorizes PI from the issuer and sends authorization response to merchant
- Authorization response message includes
  - Authorization-related info: authorization block and digital envelope
  - Capture token info: signed and encrypted token for payment, the digital envelope – assure the actual payment from the bank

Payment Authorization

- Payment capture using the capture token
  - Payment request: transmits a payment request message with its capture token
  - Payment response: receives pament