

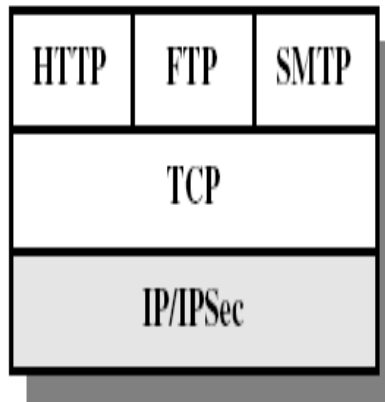
# SSL

Notes derived from Stallings book

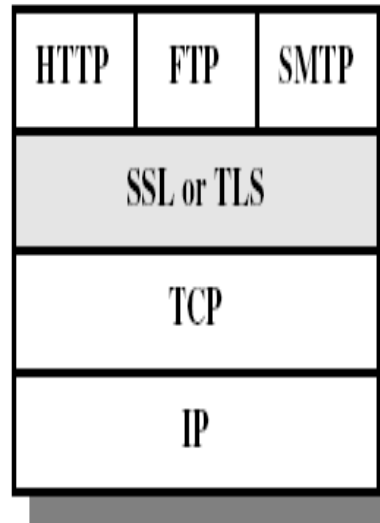
# SSL

- Secure Socket Layer
- SSL – at Transport layer (TCP)
- Netscape and I E implement this
- SSL is also known as TLS-- Transport Layer Security

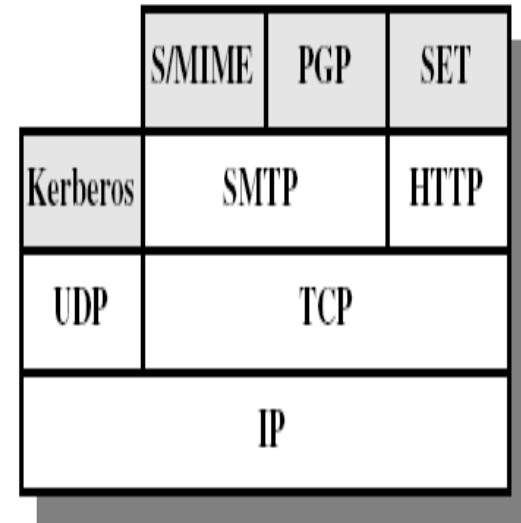
# SSL, IPSEC and others..



(a) Network Level



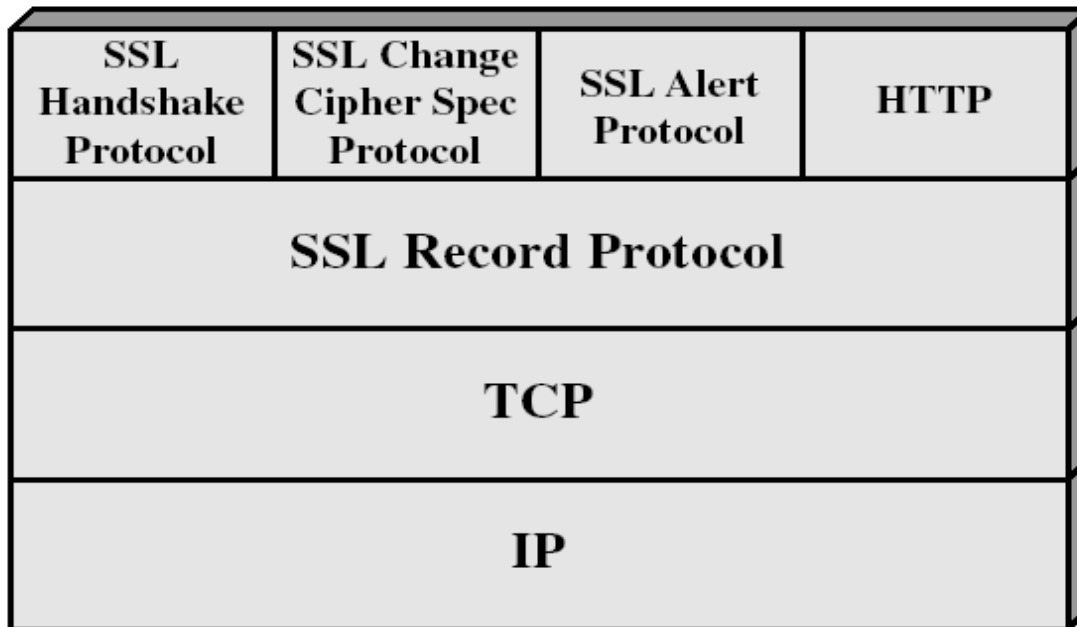
(b) Transport Level



(c) Application Level

# SSL protocols

- Consists of two layers of protocols



# SSL Protocol

- SSL Connection
  - Peer-to-peer transport
  - Transient
  - Associated with one session
- SSL Session
  - Association between client and server
  - Created by Handshake protocol
  - Defines security parameters shared across connections

# SSL Session

- Session identifier
- Peer certificate (X509.v3)
- Compression method
- Cipher Spec
  - encryption, MAC algorithms
- Master secret (48 bytes)
- Is Resumable?
  - Can we start new connections?

# SSL connection

- Server & Client Random:
  - Byte sequences chosen for each conn.
- Server write MAC Secret
  - Secret key used in MAC by server
- Client write MAC Secret
- Server Write key
  - Encryption key used by the server

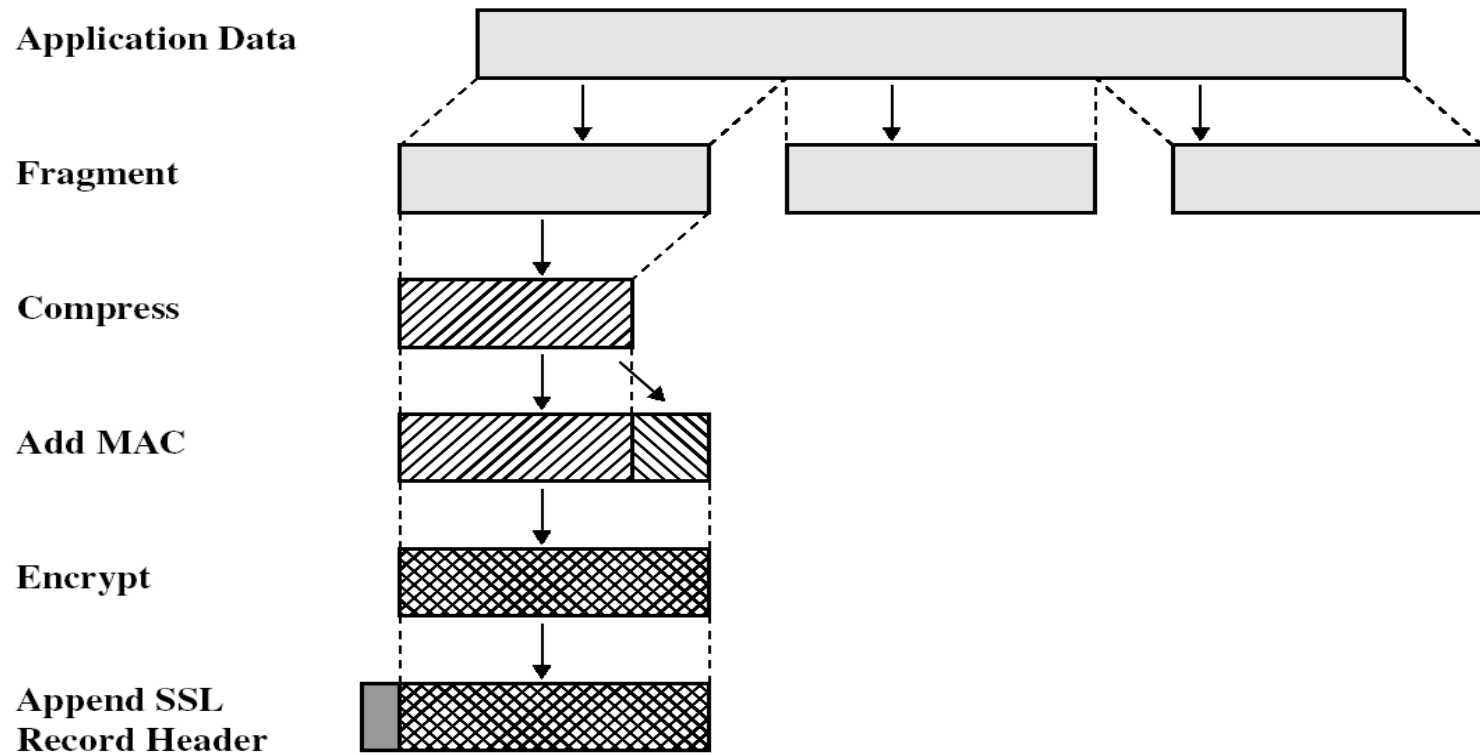
# SSL connection

- Client Write key
- Initialization vectors
  - Used in CBC mode
  - Initialized by the Handshake protocol
- Sequence numbers
  - 64 bits
  - Initialized to 0 after a change-cipher message

# SSL Record Protocol

- Provides two services
- Confidentiality through encryption
- Message Integrity through MAC

# SSL Record Protocol



# SSL Record protocol

- Fragmentation
  - Into 16Kbytes or less
- Compression
  - Optional
  - Lossless
  - Cannot increase data by more than 1kbytes

# SSL Record Protocol

- MAC very similar to HMAC
  - Concatenation used instead of XOR
- MD5 or SHA-1
- Hash(MAC\_write\_secret || pad2 || hash(MAC\_write\_secret || pad\_1 || seq\_num || SSLCompressed.type || SSLCompressed.Length || SSLCompressedfragment))

# MAC details

- MAC\_write\_secret
  - Shared secret key
- Pad\_1: byte 0x36 repeated 48 times for MD5 and 40 times for SHA-1
- Pad\_2: byte 0x5C repeated 48 times for MD5 and 40 times for SHA-1
- Seq\_num: sequence # of this message

# MAC details

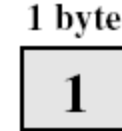
- SSLCompressed.type (8 bits)
  - Higher level protocol used to process this fragment
  - Change\_cipher\_spec, alert, handshake, application data
- Ver

# Encryption

- Block Cipher mode
  - IDEA(128), RC2-40(40), DES-40(40), DES (56), 3DES(168), Fortezza (80)
- Stream Cipher mode
  - RC4-40 (40), RC4-128 (128)

# Change Cipher Spec Protocol

- A single message
  - Single byte with value 1
- Causes pending state to be copied into current state
  - Changes the connection cipher suite



(a) Change Cipher Spec Protocol

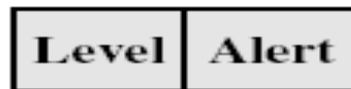
# Alert Protocol

- Used to convey alerts to SSL peer
- Two bytes
- First byte conveys the severity
  - warning
  - fatal: terminate connection
    - No new connection can be established
    - Other connections can continue
- Second byte indicates specific alert

# Alert Protocol

- SSL Alerts:
  - Unexpected message, bad\_record\_mac, decompression failure, handshake failure etc.

**1 byte 1 byte**



**(b) Alert Protocol**

# Handshake Protocol

- Allows client and server to authenticate each other
- Negotiate encryption, MAC algorithms, parameters etc.
- Used before any application data is transmitted



(c) Handshake Protocol

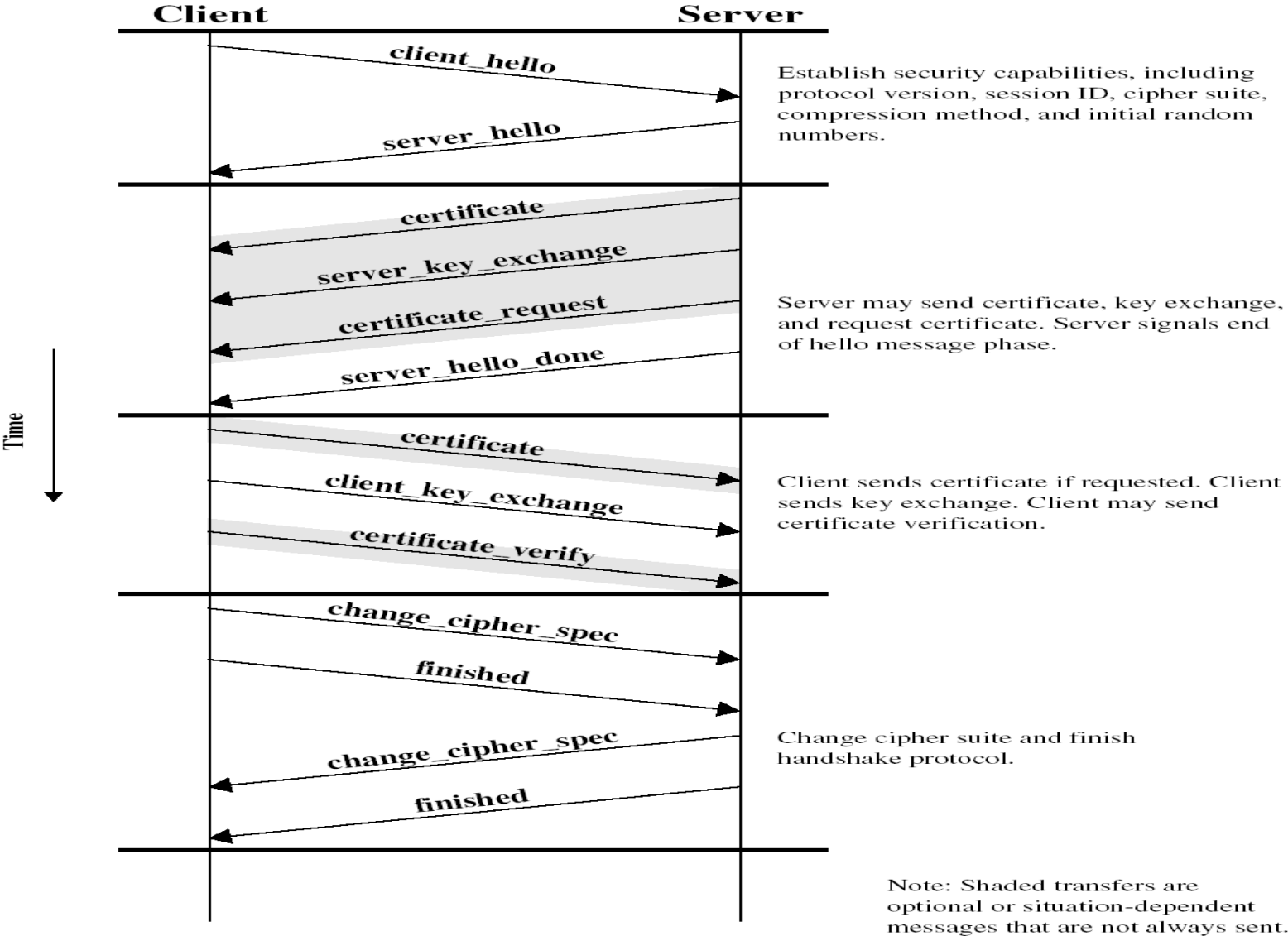
# Handshake Protocol

- Series of messages in phases
- Establish security capabilities
- Server authentication & key exchange
- Client authentication & key exchange
- Finish –proceed to data exchange

# SSL Handshake Protocol messages

| 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  |

# Handshake Protocol Action



# Phase1: Establish security capabilities

- Client initiates by a hello message
  - Highest SSL version
  - Ciphersuite –
    - all algorithms client can handle
  - Compression method -- choices
  - SessionID: variable length
    - zero: client wants to create a new connection on a new session
    - Nonzero: update parameters of current connection or new connection on current session
  - Random: nonces used during key exchange
    - Prevent replay attacks

# Server's hello

- Server picks minimum level of SSL version that client and server can support
- Server picks a Ciphersuite among the ones proposed by client
- Random field –generated by server
- SessionID: picks a new ID if needed
- CompressionMethod: picks one

# Cipher Suite

- Contains a Key exchange method
- Available options:
  - RSA
    - Encrypt with receiver's public key, public-key certificate for receiver's key needed
  - Fixed Diffie Hellman
    - Public parameters signed by a CA
  - One-time Diffie Hellman
  - Anonymous Diffie Hellman
  - Fotezza

# CipherSpec

- Cipher algorithm
  - RC4, RC2, DES, 3DES, IDEA, Fortezza
- MAC algorithm
  - MD5 or SHA-1
- Cipher Type: Stream or Block
- HashSize: 0, 16 (MD5), 20 (SHA-1)
- Key Material: used in writekey gen.
- IV Size: Initialization value of CBC

# Phase2: Server Authentication and Key Exchange

- Server sends its certificate for authentication (one or chain of CAs)
- Server key exchange message sent
  - Not required with fixed DH, RSA
- Signatures generated by encrypting  $\text{hash}(\text{ClientHello.random} || \text{ServerHello.random} || \text{ServerParams})$

# More on Phase2

- Requests a certificate from client
  - Certificate type, acceptable CAs
- Certificate types:
  - RSA, signature only, DSSsignature etc.
- Finally, sends a Serverdone message
  - Will wait for client response

# Phase 3: Client Authentication and Key Exchange

- Client verifies server's certificates
- Checks servers parameters
- Client sends a certificate message
  - Can send an alert if no suitable cert.
- Sends client-key-exchange message
  - For RSA, generates a 48-byte pre-master secret, encrypts with server's public key
  - For DH, this is null

# Phase 3

- Finally, client sends cert-verify message
  - Required for all but fixed-DH
- CertVerify.signatureMD5 =  
MD5(master\_secret || pad\_2 || MD5(handshake\_messages || master\_secret || pad\_1))

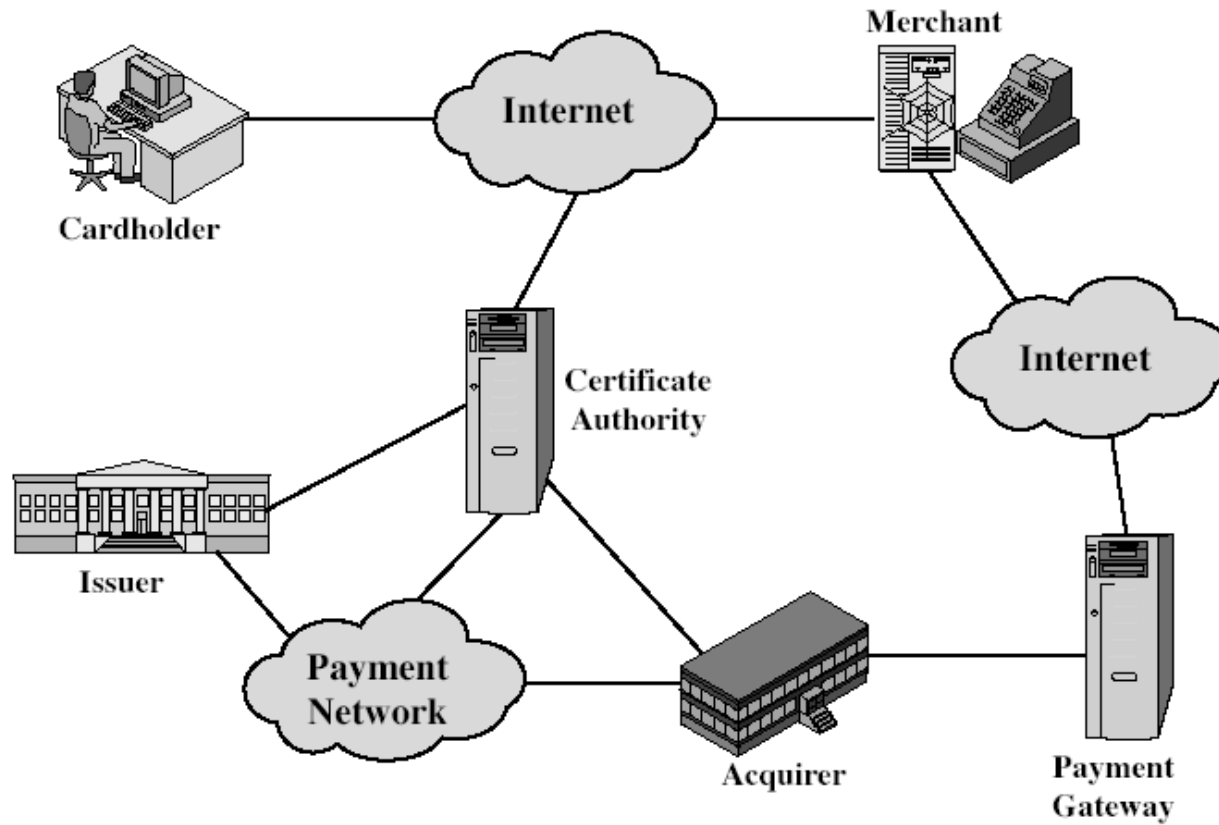
# Phase4: Finish

- Client sends change\_cipher\_spec
  - Copies pending CipherSpec into current CipherSpec
- Sends Finished message
  - Using new algorithms, keys and secrets
- Finished message concatenates MD5 and SHA hashes of
  - MD5(master\_secret||pad\_2||MD5(handshake\_messages||Sender||master\_secret||pad\_1))

# Secure Electronic Transactions

- Developed by Mastercard, Visa in 1996
- Web-based credit card transactions
- Open encryption
- A set of protocols
- Not a payment system

# SET



# SET Transaction

1. Client opens an account
2. Customer receives a certificate
  - Signed by the bank issuing the card
3. Merchants have their own certs.
  - Used in authenticating merchants
4. Customer places an Order
  - Sends an order form to the merchant

# SET Transaction

- Merchant is verified
  - Merchant returns the OI with its Cert.
  - Customer knows he is dealing with a valid merchant
- The order and payment are sent
  - Payment contains credit card details
- Merchant requests payment authorization
  - Check customer's available credit

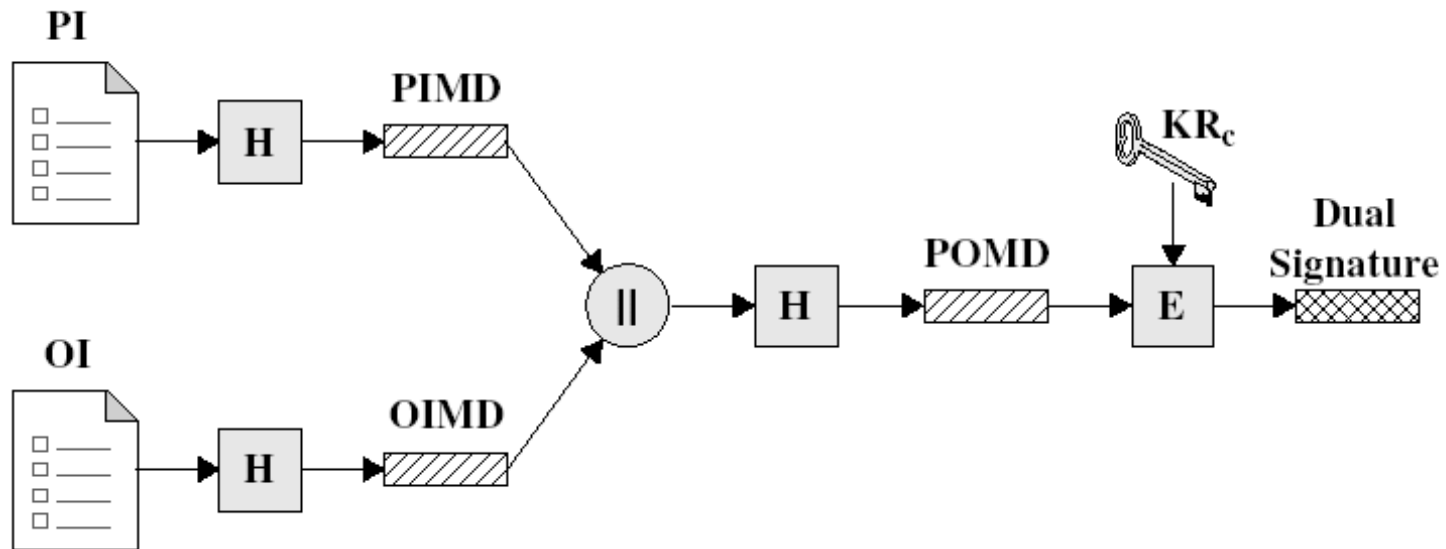
# SET Transaction

- Merchant confirms order
- Merchant provides goods or services
- Merchant requests payment

# Dual signature

- Need to sign the order and payment
- Customer needs to link these two
- Does not want to provide payment info. to the merchant
- Does not want to provide order info. to the bank -for privacy
- How?

# Dual signature



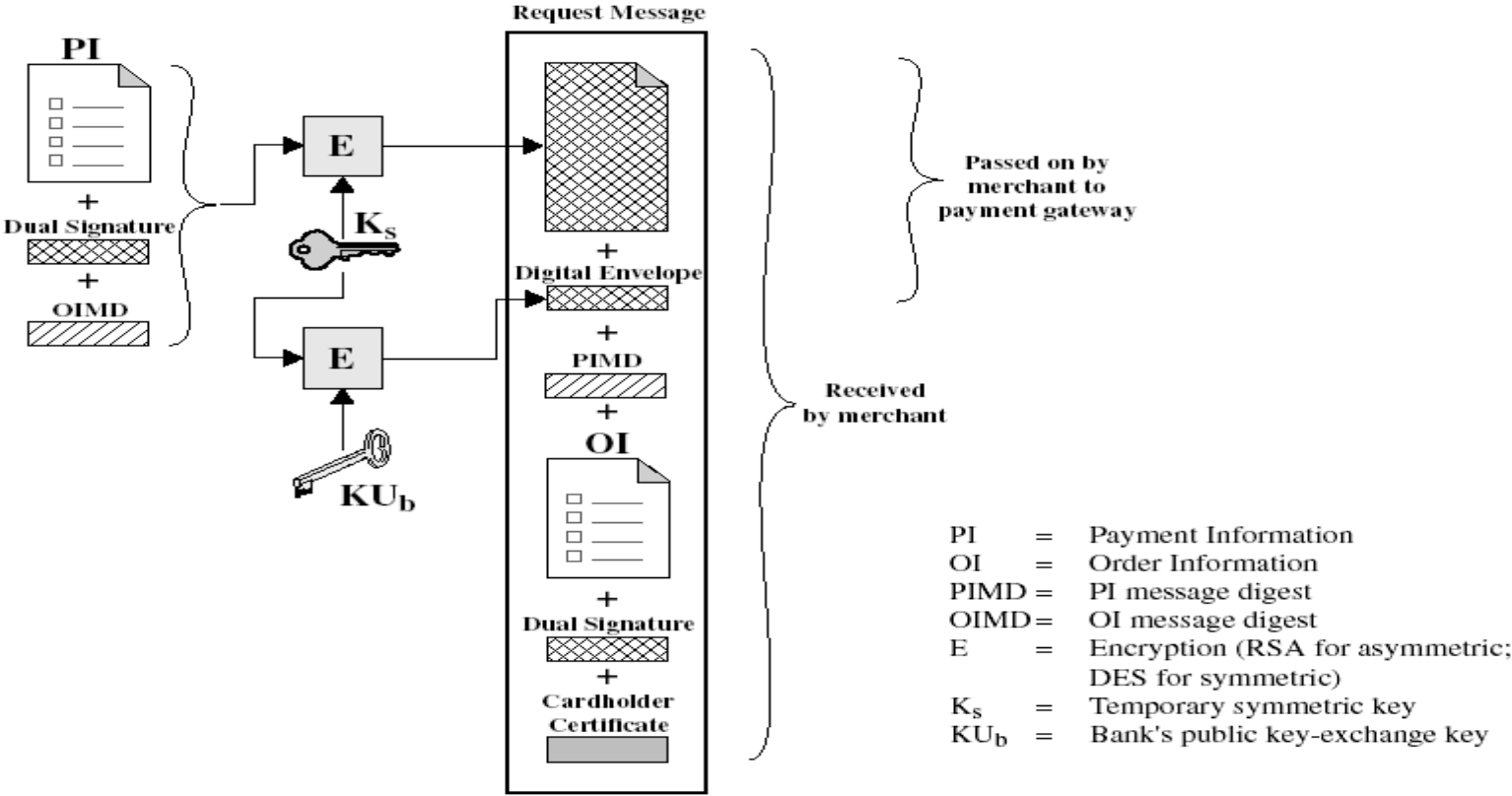
PI = Payment Information  
OI = Order Information  
H = Hash function (SHA-1)  
|| = Concatenation

PIMD = PI message digest  
OIMD = OI message digest  
POMD = Payment Order message digest  
E = Encryption (RSA)  
KR<sub>c</sub> = Customer's private signature key

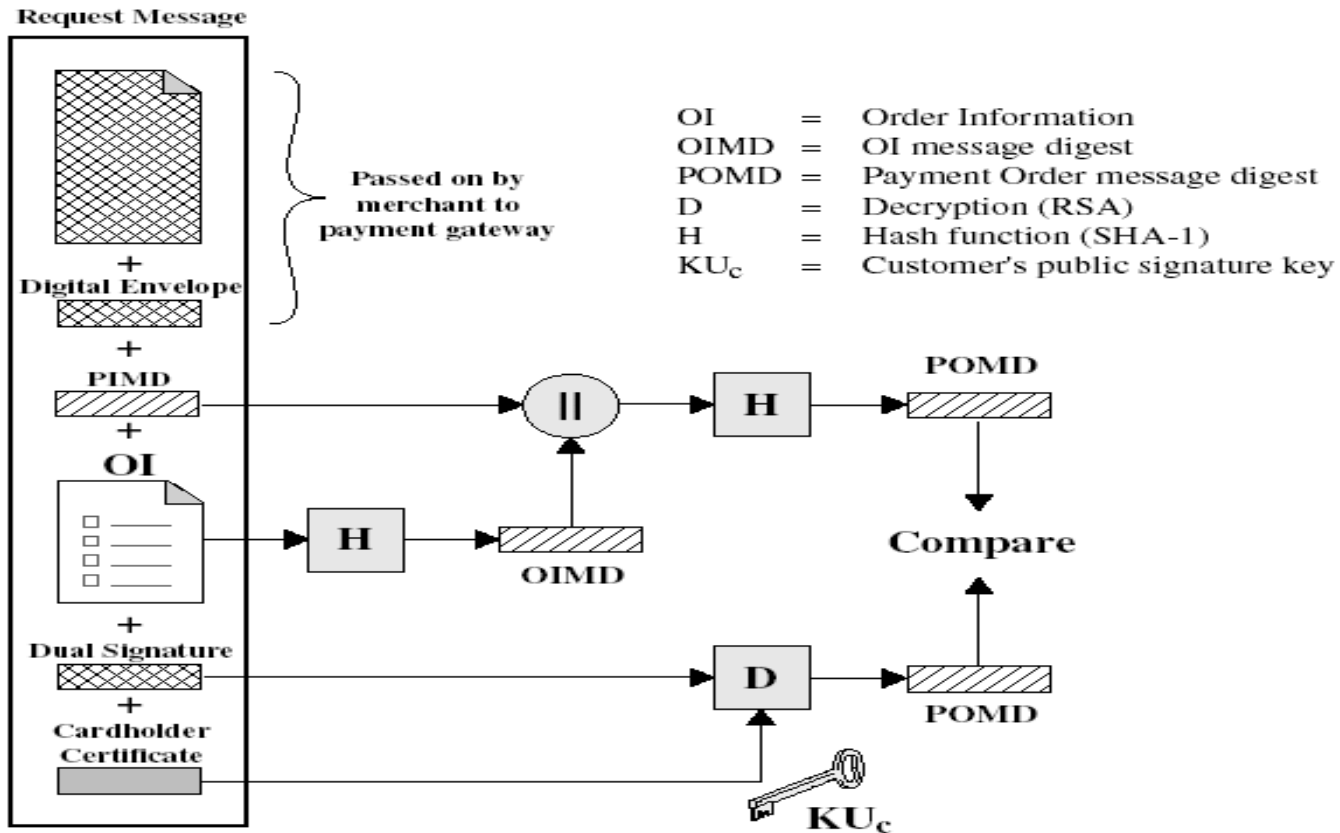
# Dual signature

- Customer generates DS by
  - $DS = E_{\text{private}}(H(H(PI) || H(OI)))$
- Merchant can verify by checking  $H(PI MD || H(OI))$  and  $D_{\text{public}}(DS)$
- Bank can verify by checking  $H(H(PI) || OI MD))$  and  $D_{\text{public}}(DS)$

# Carholder sends purchase req.



# Merchant verifies



# Want to learn more?

- Look up RFCs 2246 for TLS
- 971-page SET specification

# Summary of today's class

- SSL provides a transport layer level security architecture
- SET provides a protocol for credit card transactions on web