UNIT-IV

WEB SECURITY

CONTENT

- Requirements
- SSL
- TLS
- SET

Web Security Considerations

- The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets.
- As such, the security tools and approaches discussed so far in this book are relevant to the issue of Web security.
- But, the Web presents new challenges not generally appreciated in the context of computer and network security:
  - The Internet is two way. Web is vulnerable to attacks on the Web servers over the Internet.
  - The Web is increasingly serving as a highly visible outlet for corporate and product information and as the platform for business transactions. Reputations can be damaged and money can be lost if the Web servers are subverted.
  - Although Web browsers are very easy to use, Web servers are relatively easy to configure and manage, and Web content is increasingly easy to develop, the underlying software is extraordinarily complex. This complex software may hide many potential security flaws.

- The short history of the Web is filled with examples of new and upgraded systems, properly installed, that are vulnerable to a variety of security attacks.
- Once the Web server is subverted, an attacker may be able to gain access to data and systems not part of the Web itself but connected to the server at the local site.
- Casual and untrained (in security matters) users are common clients for Web-based services. Such users are not necessarily aware of the security risks that exist and do not have the tools or knowledge to take effective countermeasures.

Web Security Threats

- Table 1.1 provides a summary of the types of security threats faced in using the Web. One way to group these threats is in terms of passive and active attacks.
- Passive attacks include eavesdropping on network traffic between browser and server and gaining access to information on a Web site that is supposed to be restricted.
- Active attacks include impersonating another user, altering messages in transit between client and server, and altering information on a Web site.

A Comparison of Threats on the 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 Vulnerability 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

Web Traffic Security Approaches

- A number of approaches to providing Web security are possible.
- The various approaches that have been considered are similar in the services they provide and, to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack.
- One way to provide Web security is to use IP Security (Figure 1.1a). The advantage of using IPSec is that it is transparent to end users and applications and provides a general-purpose solution. Further, IPSec includes a filtering capability so that only selected traffic need incur the overhead of IPSec processing.

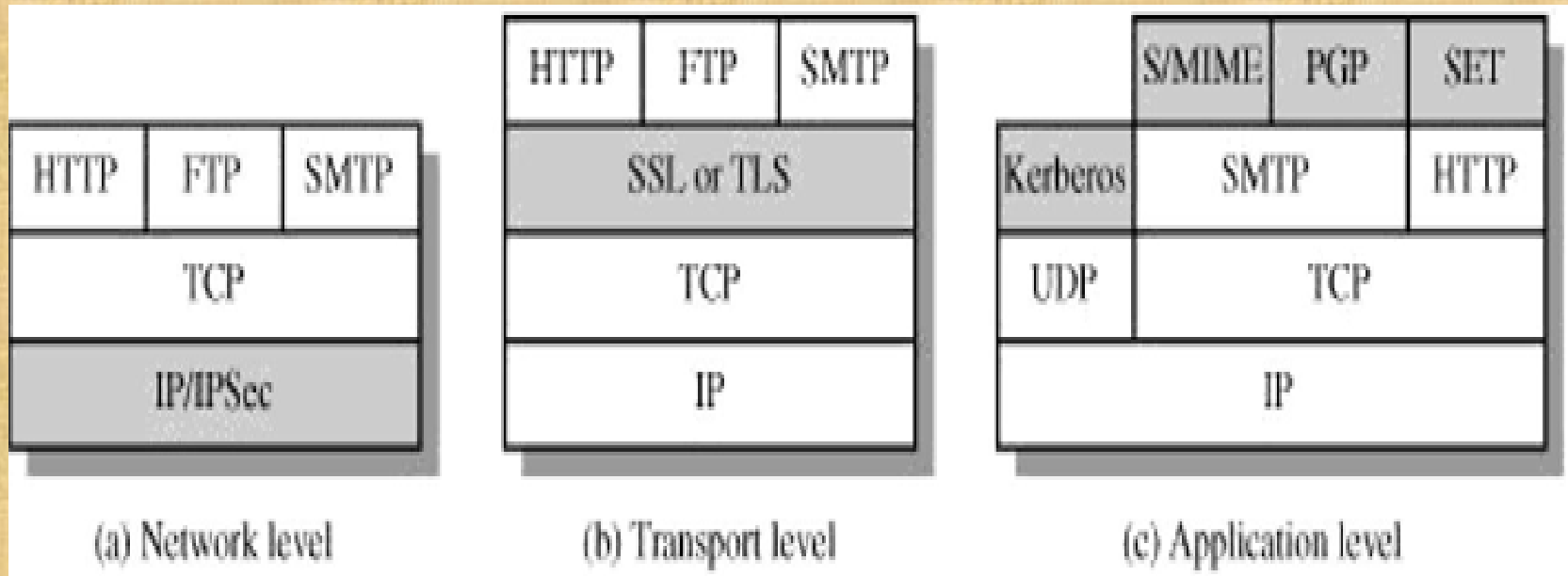


Figure: 1.1 Relative Location of Security Facilities in the TCP/IP Protocol Stack

- Another relatively general-purpose solution is to implement security just above TCP (Figure 1.1b).
- The foremost example of this approach is the Secure Sockets Layer (SSL) and the follow-on Internet standard known as Transport Layer Security (TLS).
- At this level, there are two implementation choices. For full generality, SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications.
- Alternatively, SSL can be embedded in specific packages.
- In the context of Web security, an important example of this approach is Secure Electronic Transaction (SET).

SSL Architecture

- SSL is designed to make use of TCP to provide a reliable end-to-end secure service.
- SSL is not a single protocol but rather two layers of protocols, as illustrated in Figure 5.2.
- The SSL Record Protocol provides basic security services to various higher layer protocols.
- In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL.
- **Three higher-layer protocols are defined as part of SSL: T**he Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol.
- These SSL specific protocols are used in the management of SSL exchanges and are examined later in this section.
- Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows.

- **Connection:** A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Session:** An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters which can be shared among multiple connections.
- Sessions are used to avoid the expensive negotiation of new security parameters for each connection. Between any pair of parties (applications such as HTTP on client and server), there may be multiple secure connections.
- There are a number of states associated with each session.

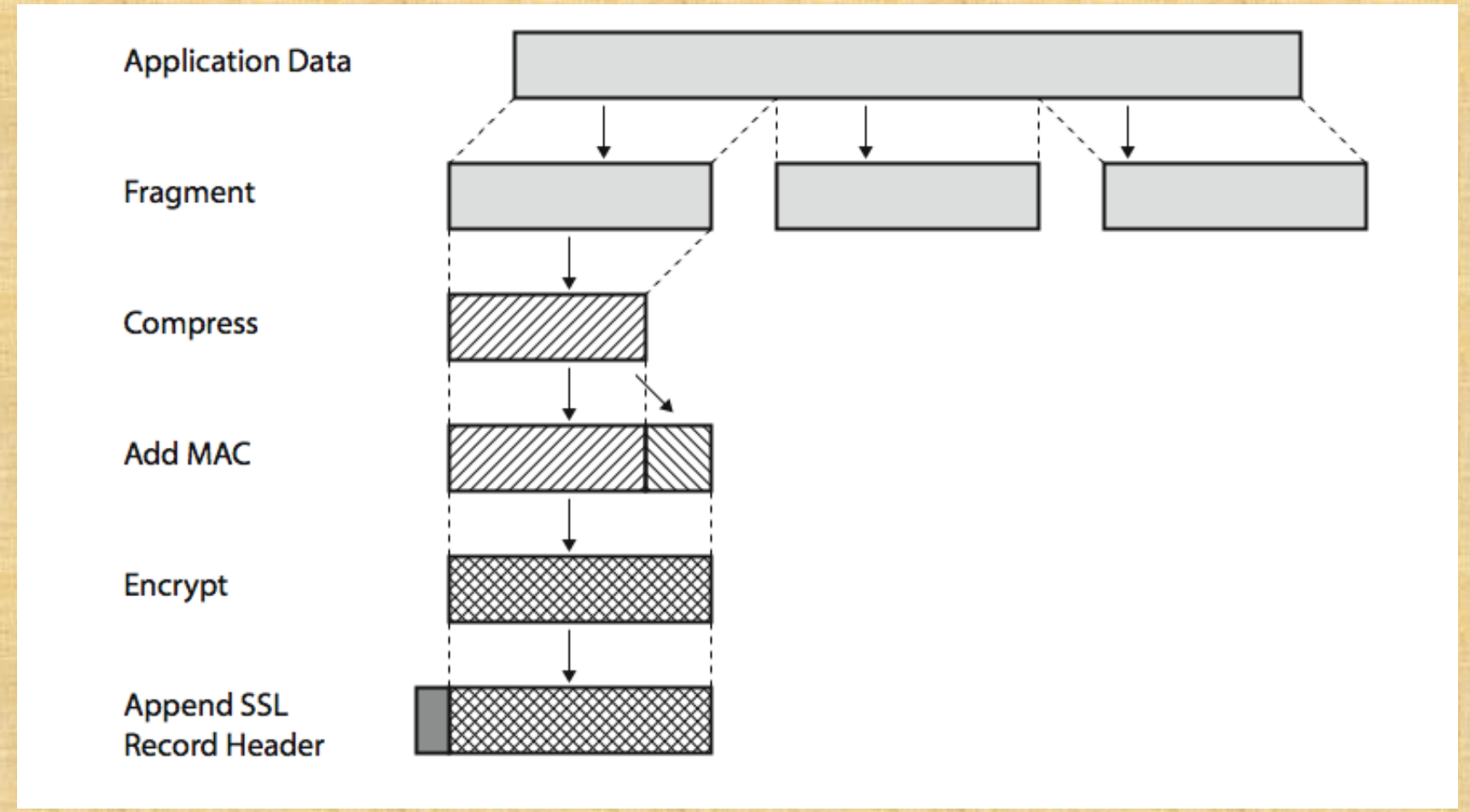
- A session state is defined by the following parameters
- **Session identifier:** An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Peer certificate: An X509.v3 certificate of the peer. This element of the state** may be null.
- **Compression method:** The algorithm used to compress data prior to encryption.
- **Cipher spec:** Specifies the bulk data encryption algorithm (such as null,AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.
- **Master secret:** 48-byte secret shared between the client and server.
- **Is resumable:** A flag indicating whether the session can be used to initiate new connections.

- A connection state is defined by the following parameters.
- **Server and client random:** Byte sequences that are chosen by the server and client for each connection.
- **Server write MAC secret:** The secret key used in MAC operations on data sent by the server.
- **Client write MAC secret:** The secret key used in MAC operations on data sent by the client.
- **Server write key:** The secret encryption key for data encrypted by the server and decrypted by the client.
- **Client write key**: The symmetric encryption key for data encrypted by the client and decrypted by the server.
- **Initialization vectors:** When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter, the final ciphertext block from each record is preserved for use as the IV with the following record.
- **Sequence numbers:** Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed 264 – 1.

SSL Record Protocol

- The SSL Record Protocol provides two services for SSL connections:
- **Confidentiality: The Handshake Protocol defines a shared secret key that is** used for conventional encryption of SSL payloads.
- **Message Integrity: The Handshake Protocol also defines a shared secret key** that is used to form a message authentication code (MAC).
- Figure 5.3 indicates the overall operation of the SSL Record Protocol.
- Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled before being delivered to higher-level users.
- The first step is **fragmentation. Each upper-layer message is fragmented** into blocks of 214 bytes (16384 bytes) or less.
- Next, **compression is optionally** applied. Compression must be lossless and may not increase the content length by more than 1024 bytes.1 In SSLv3 (as well as the current version of TLS), no compression algorithm is specified, so the default compression algorithm is null.

SSL Record Protocol Operation



- The next step in processing is to compute a **message authentication code over** the compressed data.
- For this purpose, a shared secret key is used. The calculation is defined as

hash(MAC_write_secret || pad_2|| hash(MAC_write_secret || pad_1||seq_num ||SSLCompressed.type || SSLCompressed.length || SSLCompressed.fragment))

Where, || = concatenation,

MAC_write_secret = shared secret key

hash = cryptographic hash algorithm; either MD5 or SHA-1

pad_1 = the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1

pad_2 = the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1

seq_num = the sequence number for this message

SSLCompressed.type = the higher-level protocol used to process this fragment

SSLCompressed.length = the length of the compressed fragment

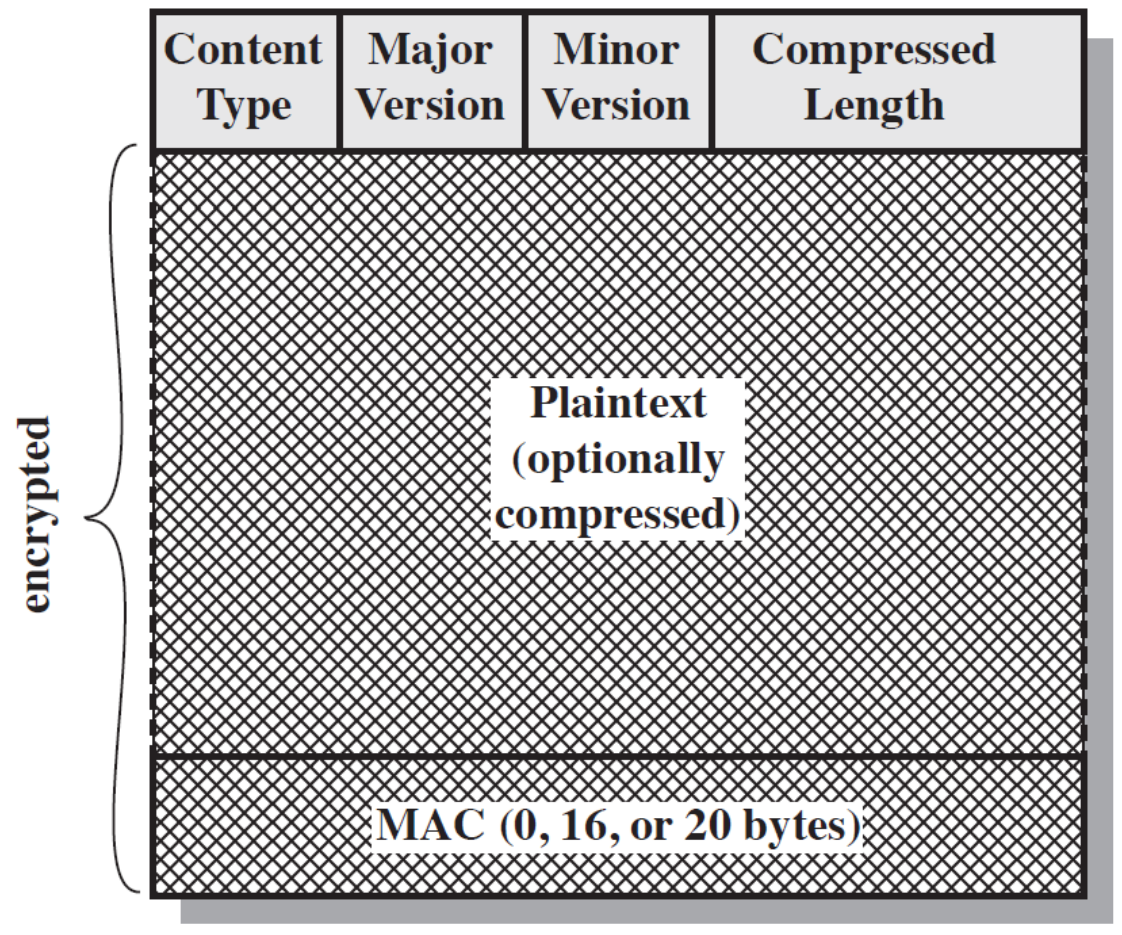
SSLCompressed.fragment = the compressed fragment (if compression is not used, this is the plaintext fragment)

- Next, the compressed message plus the MAC are **encrypted using symmetric** encryption. Encryption may not increase the content length by more than 1024 bytes, so that the total length may not exceed 214 + 2048. The following encryption algorithms are permitted:

Block Cipher		Stream Cipher	
Algorithm	Key Size	Algorithm	Key Size
AES	128, 256	RC4-40	40
IDEA	128	RC4-128	128
RC2-40	40		
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

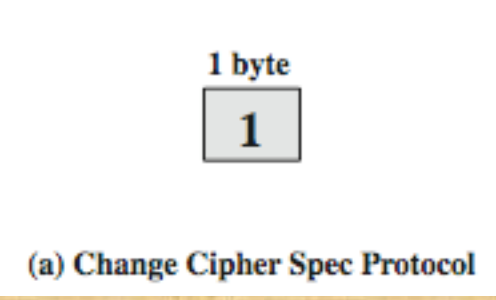
- The final step of SSL Record Protocol processing is to prepare a header consisting of the following fields:
- **Content Type (8 bits): The higher-layer protocol used to process the enclosed** fragment.
- **Major Version (8 bits): Indicates major version of SSL in use. For SSLv3, the** value is 3.
- **Minor Version (8 bits): Indicates minor version in use. For SSLv3, the value is 0.**
- **Compressed Length (16 bits): The length in bytes of the plaintext fragment (or** compressed fragment if compression is used).The maximum value is 2^14+2048
- The content types that have been defined are change_cipher_spec, alert, handshake, and application_data. The first three are the SSL-specific protocols, discussed next.

SSL Record Format



Change Cipher Spec Protocol

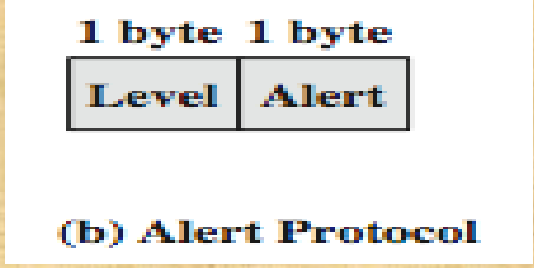
- The Change Cipher Spec Protocol is one of the three SSL-specific protocols that use the SSL Record Protocol, and it is the simplest.
- This protocol consists of a single message (Figure 5.5a), which consists of a single byte with the value 1.
- The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.



(a) Change Cipher Spec Protocol

Alert Protocol

- The Alert Protocol is used to convey SSL-related alerts to the peer entity.
- As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current state.
- Each message in this protocol consists of two bytes (Figure 5.5b).The first byte takes the value warning (1) or fatal (2) to convey the severity of the message.
- If the level is fatal, SSL immediately terminates the connection. Other connections on the same session may continue, but no new connections on this session may be established.
- The second byte contains a code that indicates the specific alert. First, we list those alerts that are always fatal (definitions from the SSL specification):



- **unexpected_message: An inappropriate message was received.**
- **bad_record_mac: An incorrect MAC was received.**
- **decompression_failure: The decompression function received improper** input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake_failure: Sender was unable to negotiate an acceptable set of** security parameters given the options available.
- **illegal_parameter: A field in a handshake message was out of range or** inconsistent with other fields. The remaining alerts are the following.
- **close_notify: Notifies the recipient that the sender will not send any** more messages on this connection. Each party is required to send a
- **close_notify alert before closing the write side of a connection.**
- **no_certificate: May be sent in response to a certificate request if no** appropriate certificate is available.
- **bad_certificate: A received certificate was corrupt (e.g., contained a** signature that did not verify).
- **unsupported_certificate: The type of the received certificate is not** supported.
- **certificate_revoked: A certificate has been revoked by its signer.**
- **certificate_expired: A certificate has expired.**
- **certificate_unknown: Some other unspecified issue arose in processing** the

Handshake Protocol

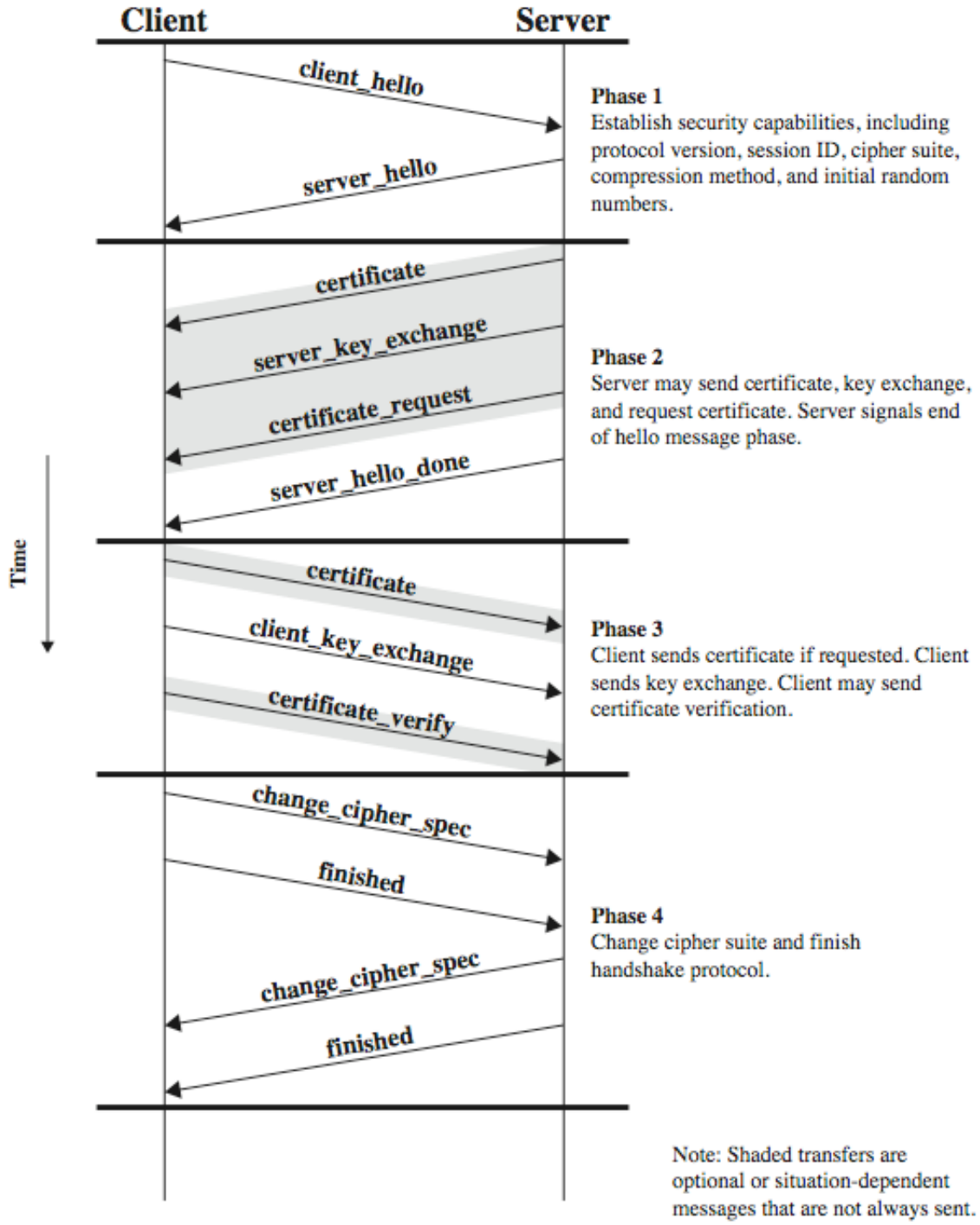
- The most complex part of SSL is the Handshake Protocol.
- This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in an SSL record.
- The Handshake Protocol is used before any application data is transmitted.
- The Handshake Protocol consists of a series of messages exchanged by client and server.
- All of these have the format shown in Figure 5.5c. Each message has three fields:
  - **Type (1 byte): Indicates one of 10 messages. Table 5.2 lists the defined** message types.
  - **Length (3 bytes): The length of the message in bytes.**
  - **Content (bytes): The parameters associated with this message; these are** listed in Table 5.2.

PHASE 1. ESTABLISH SECURITY CAPABILITIES

Table 5.2 SSL Handshake Protocol 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 Protocol



- This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it.
- The exchange is initiated by the client, which sends a client_hello message with the following parameters:
- **Version:** The highest SSL version understood by the client.
- **Random:** A client-generated random structure consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.
- **Session ID:** A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or to create a new connection on this session. A zero value indicates that the client wishes to establish a new connection on a new session.
- **CipherSuite:** This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec; these are discussed subsequently.
- **Compression Method:** This is a list of the compression methods the client supports.

- The first element of the CipherSuite parameter is the key exchange method. The following key exchange methods are supported.
- **RSA: The secret key is encrypted with the receiver's RSA public key.A publickey** certificate for the receiver's key must be made available.
- **Fixed Diffie-Hellman: This is a Diffie-Hellman key exchange in which the**
- server's certificate contains the Diffie-Hellman public parameters signed by
- the certificate authority (CA). That is, the public-key certificate contains the
- Diffie-Hellman public-key parameters. The client provides its Diffie-Hellman
- public-key parameters either in a certificate, if client authentication is
- required, or in a key exchange message. This method results in a fixed secret
- key between two peers based on the Diffie-Hellman calculation using the
- fixed public keys.

- **RSA: The secret key is encrypted with the receiver's RSA public key.A publickey** certificate for the receiver's key must be made available.
- **Fixed Diffie-Hellman: This is a Diffie-Hellman key exchange in which the** server's certificate contains the Diffie-Hellman public parameters signed by the certificate authority (CA). That is, the public-key certificate contains the Diffie-Hellman public-key parameters. The client provides its Diffie-Hellman public-key parameters either in a certificate, if client authentication is required, or in a key exchange message. This method results in a fixed secret key between two peers based on the Diffie-Hellman calculation using the fixed public keys.
- **Ephemeral Diffie-Hellman: This technique is used to create ephemeral** (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key. The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys. This would appear to be the most secure of the three Diffie-Hellman options, because it results in a temporary, authenticated key.
- **Anonymous Diffie-Hellman: The base Diffie-Hellman algorithm is used with** no authentication.That is, each side sends its public Diffie-Hellman parameters to the other with no authentication.This approach is vulnerable to man-in-themiddle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.
- **Fortezza: The technique defined for the Fortezza scheme.**

TRANSPORT LAYER SECURITY

- TLS is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL.TLS is defined as a Proposed Internet Standard in RFC 5246. RFC 5246 is very similar to SSLv3. In this section, we highlight the differences.
- Version Number The TLS Record Format is the same as that of the SSL Record Format (Figure5.4), and the fields in the header have the same meanings. The one difference is in version values. For the current version of TLS, the major version is 3 and the minor version is 3.

Message Authentication Code

- There are two differences between the SSLv3 and TLS MAC schemes: the actual algorithm and the scope of the MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104.Recall from Chapter 3 that HMAC is defined as

HMAC K (M)= H[(K + opad)||H[(K + ipad)|| M]]

where

H = embedded hash function (for TLS,either MD5 or SHA-1)

M = message input to HMAC

K + = secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1,block length = 512 bits)

ipad = 00110110 (36 in hexadecimal) repeated 64 times (512 bits)

opad = 01011100 (5C in hexadecimal) repeated 64 times (512 bits)

- SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. The level of security should be about the same in both cases.

- For TLS, the MAC calculation encompasses the fields indicated in the following expression:

MAC(MAC_write_secret,seq_num || TLSCompressed.type || TLSCompressed.version || TLSCompressed.length || TLSCompressed.fragment)

- The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field TLSCompressed.version, which is the version of the protocol being employed.

Pseudorandom Function

- TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation.
- The objective is to make use of a relatively small shared secret value but to generate longer blocks of data in a way that is secure from the kinds of attacks made on hash functions and MACs.
- The PRF is based on the data expansion function (Figure 5.7) given as

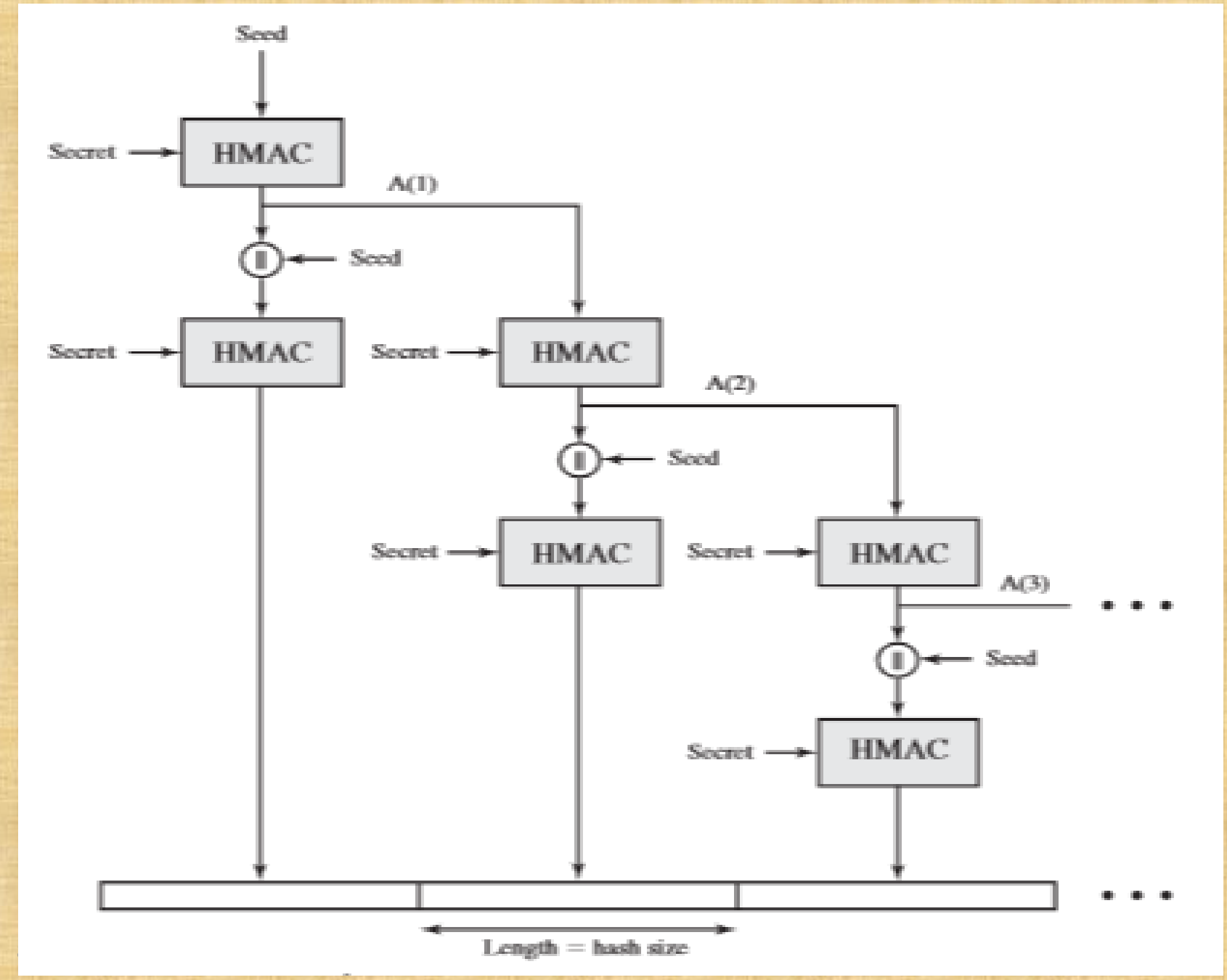
P_hash(secret, seed)= HMAC_hash(secret,A(1) || seed) ||
HMAC_hash(secret, A(2) || seed) || HMAC_hash(secret, A(3) || seed) || . . .

where

A() is defined as

A(0)= seed

A(i)= HMAC_hash(secret,A(i – 1))


Seed
Secret
HMAC
A(1)
Seed
Secret
HMAC
Secret
HMAC
A(2)
Seed
Secret
HMAC
Secret
HMAC
A(3)
Seed
Secret
HMAC
Length = hash size

Alert Codes

- TLS supports all of the alert codes defined in SSLv3 with the exception of no_certificate. A number of additional codes are defined in TLS;of these,the following are always fatal.
- record_overflow: A TLS record was received with a payload (ciphertext) whose length exceeds bytes,or the ciphertext decrypted to a length of greater than bytes.
- unknown_ca: A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known,trusted CA.
- access_denied: A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation.
- decode_error: A message could not be decoded, because either a field was out of its specified range or the length of the message was incorrect.

- protocol_version: The protocol version the client attempted to negotiate is recognized but not supported.
- insufficient_security: Returned instead of handshake_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- unsupported_extension: Sent by clients that receive an extended server hello containing an extension not in the corresponding client hello.
- internal_error: An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue.
- decrypt_error: A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange, or validate a finished message.
- user_canceled: This handshake is being canceled for some reason unrelated to a protocol failure.
- no_renegotiation: Sent by a client in response to a hello request or by the server in response to a client hello after initial handshaking.
- Either of these messages would normally result in renegotiation, but this alert indicates that the sender is not able to renegotiate.
- This message is always a warning.

Cipher Suites

- There are several small differences between the cipher suites available under SSLv3 and under TLS:
- Key Exchange: TLS supports all of the key exchange techniques of SSLv3 with the exception of Fortezza.
- Symmetric Encryption Algorithms: TLS includes all of the symmetric encryption algorithms found in SSLv3,with the exception of Fortezza.

Client Certificate Types

- TLS defines the following certificate types to be requested in a certificate_request message: rsa_sign, dss_sign, rsa_fixed_dh, and dss_fixed_dh.
- These are all defined in SSLv3.
- In addition, SSLv3 includes rsa_ephemeral_dh, dss_ephemeral_dh, and fortezza_kea.
- Ephemeral Diffie-Hellman involves signing the Diffie-Hellman parameters with either RSA or DSS.
- For TLS, the rsa_sign and dss_sign types are used for that function; a separate signing type is not needed to sign Diffie-Hellman parameters.
- TLS does not include the Fortezza scheme.

TLS (Transport Layer Security)

- IETF standard RFC 2246 similar to SSLv3
- with minor differences
  - in record format version number
  - uses HMAC for MAC
  - a pseudo-random function expands secrets
  - has additional alert codes
  - some changes in supported ciphers
  - changes in certificate negotiations
  - changes in use of padding

Secure Electronic Transaction

(SET)

Credit Cards on the Internet

- Problem: communicate credit card and purchasing data securely to gain consumer trust
  - Authentication of buyer and merchant
  - Confidential transmissions
- Systems vary by
  - Type of public-key encryption
  - Type of symmetric encryption
  - Message digest algorithm
  - Number of parties having private keys
  - Number of parties having certificates

Credit Card Protocols

- SSL 1 or 2 parties have private keys
- TLS (Transport Layer Security)
  - IETF version of SSL

- *i*KP (IBM)
- SEPP (Secure Encryption Payment Protocol)
  - MasterCard, IBM, Netscape
- STT (Secure Transaction Technology)
  - VISA, Microsoft

OBSOLETE

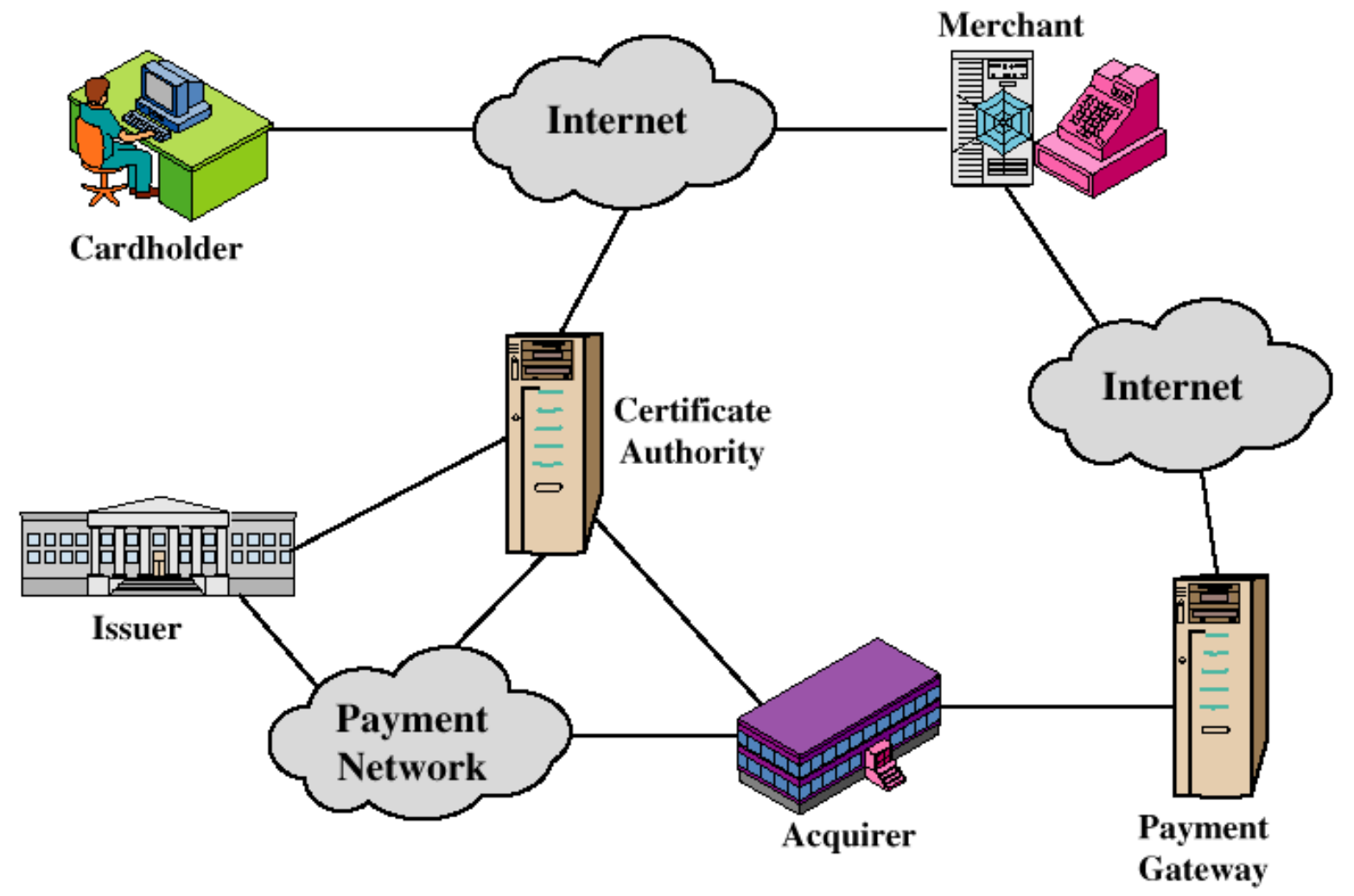
- SET (Secure Electronic Transactions)
  - MasterCard, VISA all parties have certificates

VERY SLOW ACCEPTANCE

Secure Electronic Transaction (SET)

- Developed by Visa and MasterCard
- Designed to protect credit card transactions
- Confidentiality: all messages encrypted
- Trust: all parties must have digital certificates
- Privacy: information made available only when and where necessary

Participants in the SET System



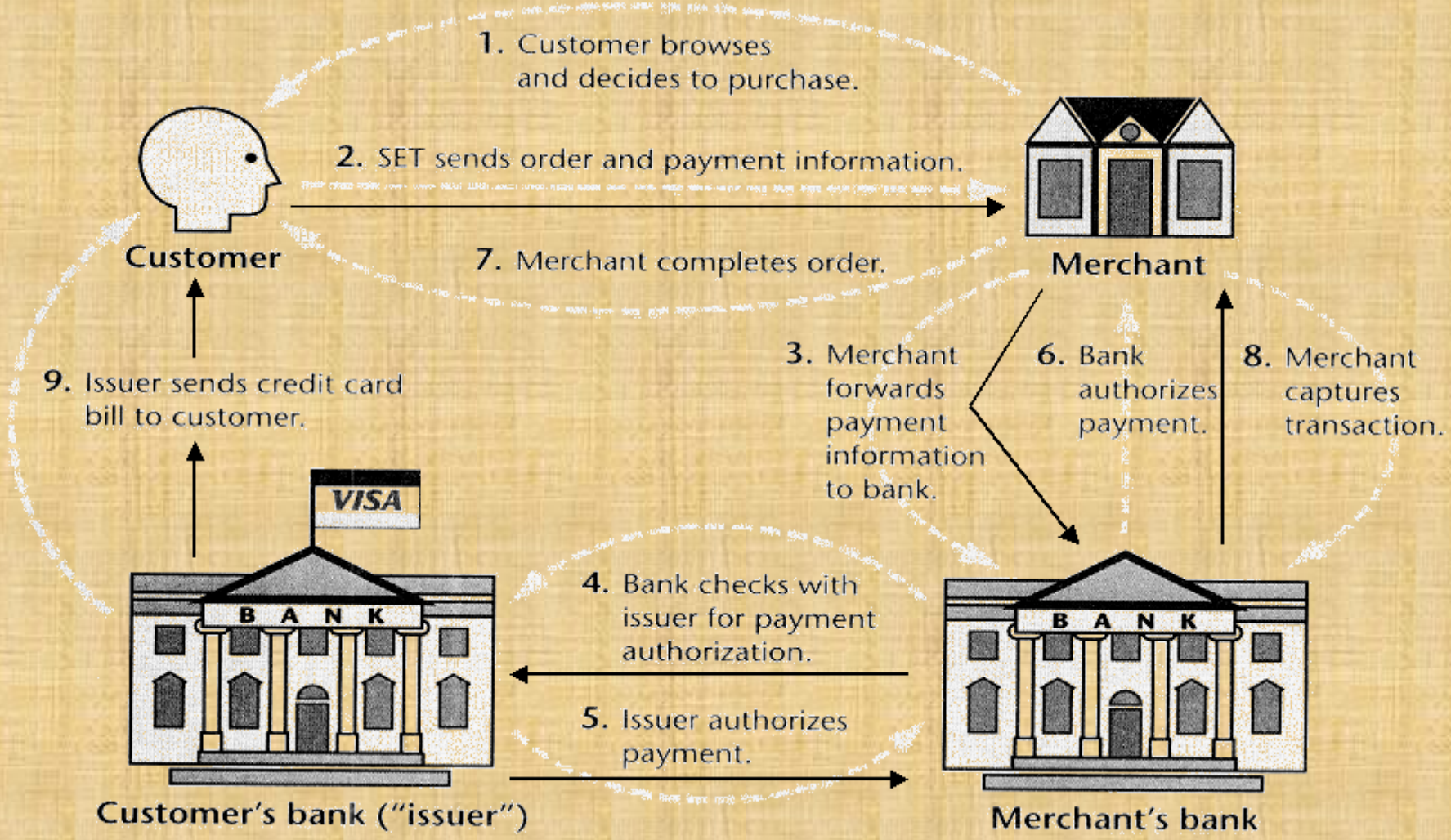
SET Business Requirements

- Provide confidentiality of payment and ordering information
- Ensure the integrity of all transmitted data
- Provide authentication that a cardholder is a legitimate user of a credit card account
- Provide authentication that a merchant can accept credit card transactions through its relationship with a financial institution

SET Business Requirements (cont'd)

- Ensure the use of the best security practices and system design techniques to protect all legitimate parties in an electronic commerce transaction
- Create a protocol that neither depends on transport security mechanisms nor prevents their use
- Facilitate and encourage interoperability among software and network providers

SET Transactions



SET Transactions

- The customer opens an account with a card issuer.
  - MasterCard, Visa, etc.
- The customer receives a X.509 V3 certificate signed by a bank.
  - X.509 V3
- A merchant who accepts a certain brand of card must possess two X.509 V3 certificates.
  - One for signing & one for key exchange
- The customer places an order for a product or service with a merchant.
- The merchant sends a copy of its certificate for verification.

SET Transactions

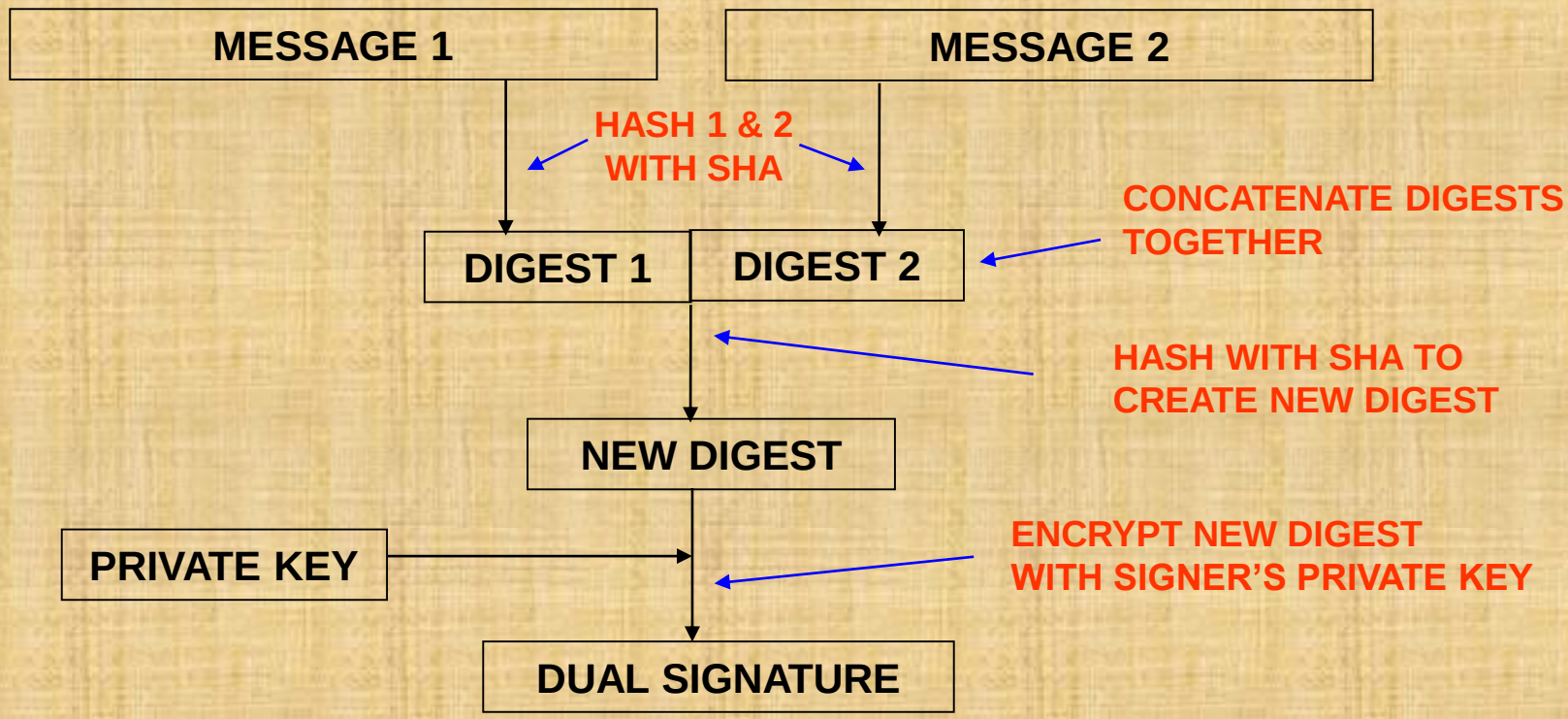
- The customer sends order and payment information to the merchant.
- The merchant requests payment authorization from the payment gateway prior to shipment.
- The merchant confirms order to the customer.
- The merchant provides the goods or service to the customer.
- The merchant requests payment from the payment gateway.

Key Technologies of SET

- Confidentiality of information: DES
- Integrity of data: RSA digital signatures with SHA-1 hash codes
- Cardholder account authentication: X.509v3 digital certificates with RSA signatures
- Merchant authentication: X.509v3 digital certificates with RSA signatures
- Privacy: separation of order and payment information using dual signatures

Dual Signatures

- Links two messages securely but allows only one party to read each.



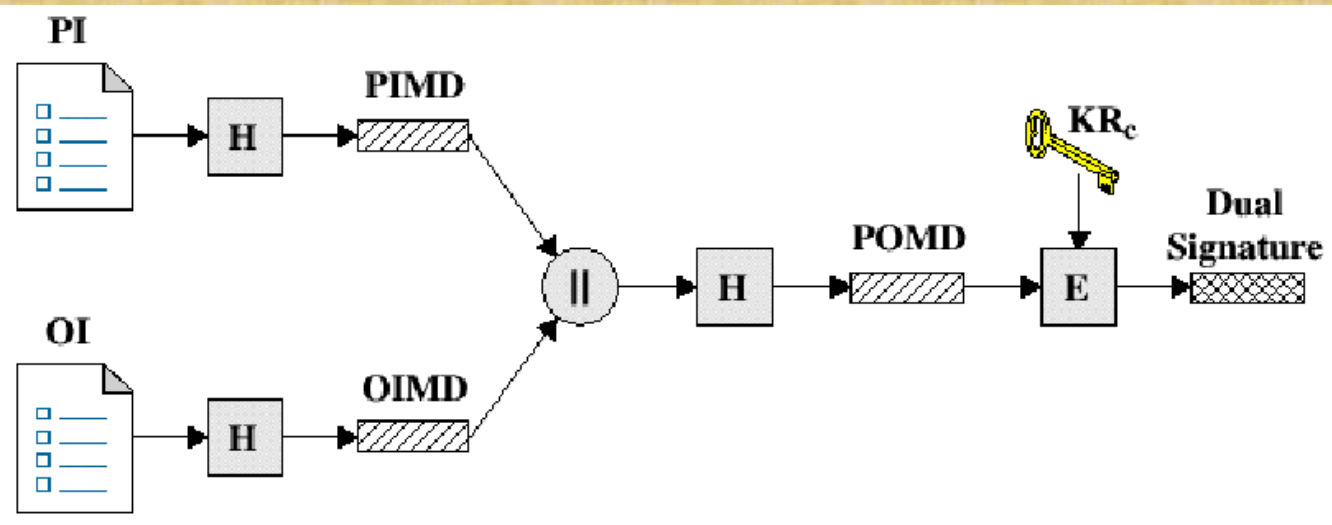
Dual Signature for SET

- Concept: Link Two Messages Intended for Two Different Receivers:
  - Order Information (OI): Customer to Merchant
  - Payment Information (PI): Customer to Bank
- Goal: Limit Information to A "Need-to-Know" Basis:
  - Merchant does not need credit card number.
  - Bank does not need details of customer order.
  - Afford the customer extra protection in terms of privacy by keeping these items separate.
- This link is needed to prove that payment is intended for this order and not some other one.

Why Dual Signature?

- Suppose that customers send the merchant two messages:
  - The signed order information (OI).
  - The signed payment information (PI).
  - In addition, the merchant passes the payment information (PI) to the bank.
- If the merchant can capture another order information (OI) from this customer, the merchant could claim this order goes with the payment information (PI) rather than the original.

Dual Signature Operation



- The operation for dual signature is as follows:
  - Take the hash (SHA-1) of the payment and order information.
  - These two hash values are concatenated [H(PI) || H(OI)] and then the result is hashed.
  - Cu**stomer encrypts the final hash with a private key creating the dual signature**.

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

DS Verification by Merchant

- The merchant has the public key of the customer obtained from the customer's certificate.
- Now, the merchant can compute two values:

  $H(PIMD \mid\mid H(OI))$

  $D_{KUC}[DS]$
- Should be equal!

DS Verification by Bank

- The bank is in possession of DS, PI, the message digest for OI (OIMD), and the customer's public key, then the bank can compute the following:

  H(H(PI) || OIMD)

  D_{KUC} [DS]

What did we accomplish?

- The merchant has received OI and verified the signature.
- The bank has received PI and verified the signature.
- The customer has linked the OI and PI and can prove the linkage.

SET Supported Transactions

- card holder registration
- merchant registration
- purchase request
- payment authorization
- payment capture
- certificate query
- purchase inquiry
- **purchase notification**
- **sale transaction**
- **authorization reversal**
- **capture reversal**
- **credit reversal**

Purchase Request

- Browsing, Selecting, and Ordering is Done
- Purchasing Involves 4 Messages:
  - Initiate Request
  - Initiate Response
  - Purchase Request
  - Purchase Response

Purchase Request: Initiate Request

- Basic Requirements:
  - Cardholder Must Have Copy of Certificates for Merchant and Payment Gateway
- Customer Requests the Certificates in the Initiate Request Message to Merchant
  - Brand of Credit Card
  - ID Assigned to this Request/response pair by customer
  - Nonce

Purchase Request: Initiate Response

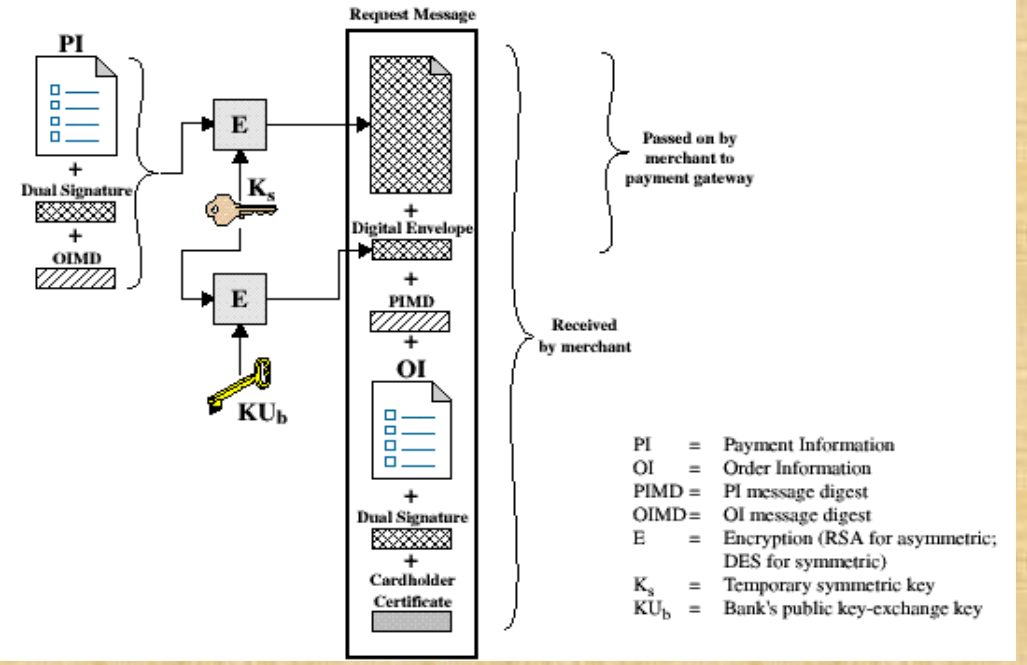
- Merchant Generates a Response
  - Signs with Private Signature Key
  - Include Customer Nonce
  - Include Merchant Nonce (Returned in Next Message)
  - Transaction ID for Purchase Transaction
- In Addition …
  - Merchant's Signature Certificate
  - Payment Gateway's Key Exchange Certificate

Purchase Request: Purchase Request

- Cardholder Verifies Two Certificates Using Their CAs and Creates the OI and PI.
- Message Includes:
  - Purchase-related Information
  - Order-related Information
  - Cardholder Certificate

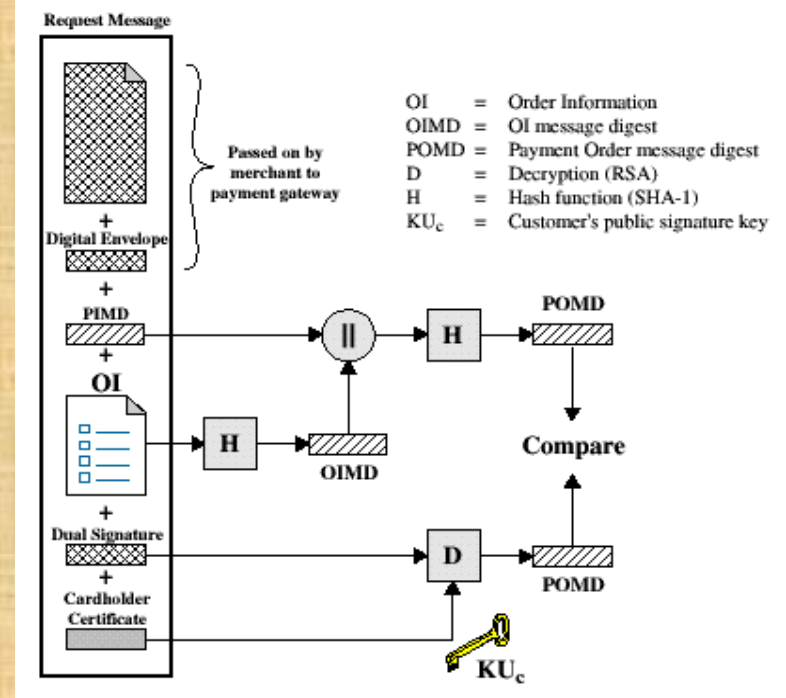
Purchase Request

- The cardholder generates a one-time symmetric encryption key, KS,



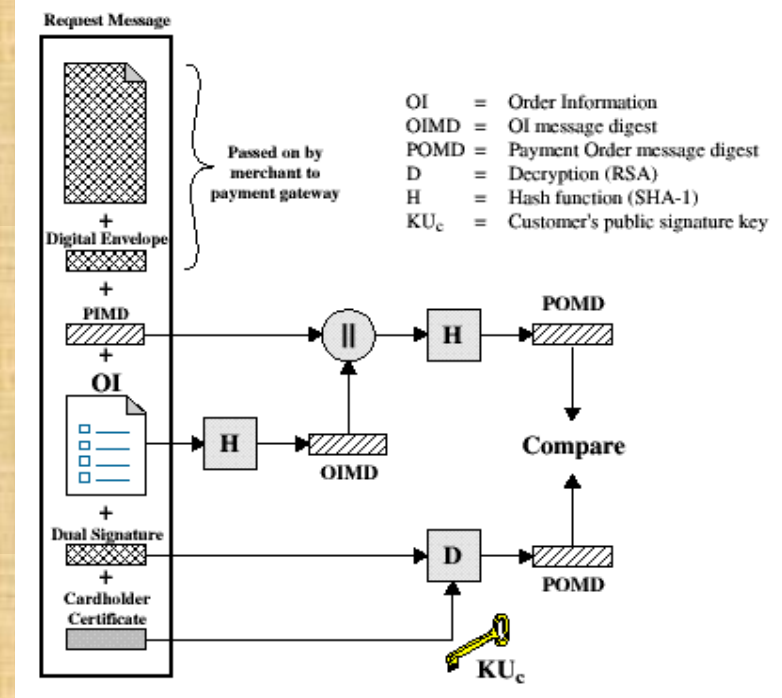
Merchant Verifies Purchase Request

- When the merchant receives the Purchase Request message, it performs the following actions:
  - Verify the cardholder certificates by means of its CA signatures.
  - Verifies the dual signature using the customer's public key signature.



Merchant Verification (cont'd)

- Processes the order and forwards the payment information to the payment gateway for authorization.
- Sends a purchase response to the cardholder.



Purchase Response Message

- Message that Acknowledges the Order and References Corresponding Transaction Number
- Block is
  - Signed by Merchant Using its Private Key
  - Block and Signature Are Sent to Customer Along with Merchant's Signature Certificate
- Upon Reception
  - Verifies Merchant Certificate
  - Verifies Signature on Response Block
  - Takes the Appropriate Action

Payment Process

- The payment process is broken down into two steps:
  - Payment authorization
  - Payment capture

Payment Authorization

- The merchant sends an authorization request message to the payment gateway consisting of the following:
  - Purchase-related information
    - PI
    - Dual signature calculated over the PI & OI and signed with customer's private key.
    - The OI message digest (OIMD)
    - The digital envelop
  - Authorization-related information
  - Certificates

Payment Authorization (cont'd)

- Authorization-related information
  - An authorization block including:
    - A transaction ID
    - Signed with merchant's private key
    - Encrypted one-time session key
- Certificates
  - Cardholder's signature key certificate
  - Merchant's signature key certificate
  - Merchant's key exchange certificate

Payment: Payment Gateway

- Verify All Certificates
- Decrypt Authorization Block Digital Envelope to Obtain Symmetric Key and Decrypt Block
- Verify Merchant Signature on Authorization Block
- Decrypt Payment Block Digital Envelope to Obtain Symmetric Key and Decrypt Block
- Verify Dual Signature on Payment Block
- Verify Received Transaction ID Received from Merchant Matches PI Received from Customer
- Request and Receive Issuer Authorization

Authorization Response

- Authorization Response Message
  - Authorization-related Information
  - Capture Token Information
  - Certificate

SET Overhead

Simple purchase transaction:

- Four messages between merchant and customer
- Two messages between merchant and payment gateway
- 6 digital signatures
- 9 RSA encryption/decryption cycles
- 4 DES encryption/decryption cycles
- 4 certificate verifications

Scaling:

- Multiple servers need copies of all certificates