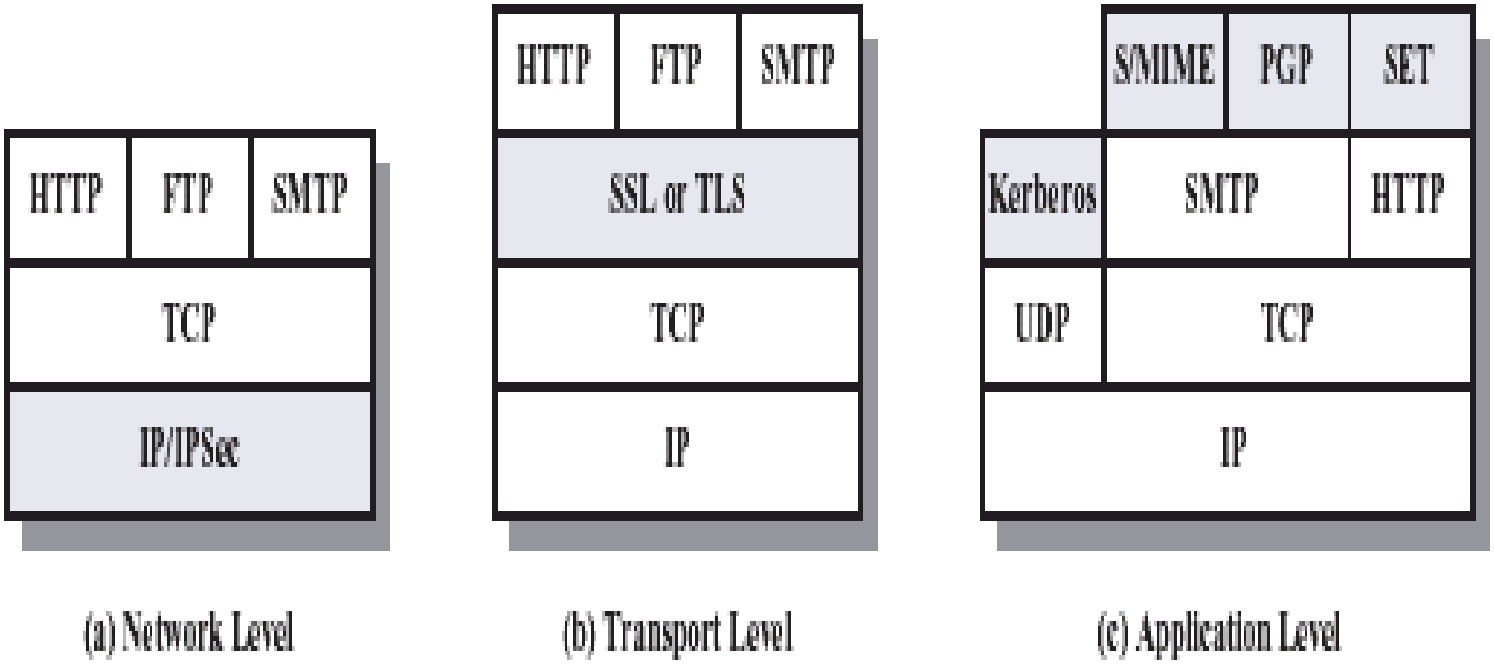
Network Security
18CSE354T

Unit 4 Notes

Prepared by:
Sonam Singh
(Course Cordinator)

SLO - 1 & 2 :
SSL/TLS BASIC PROTOCOL

Layers of Security



SSL History

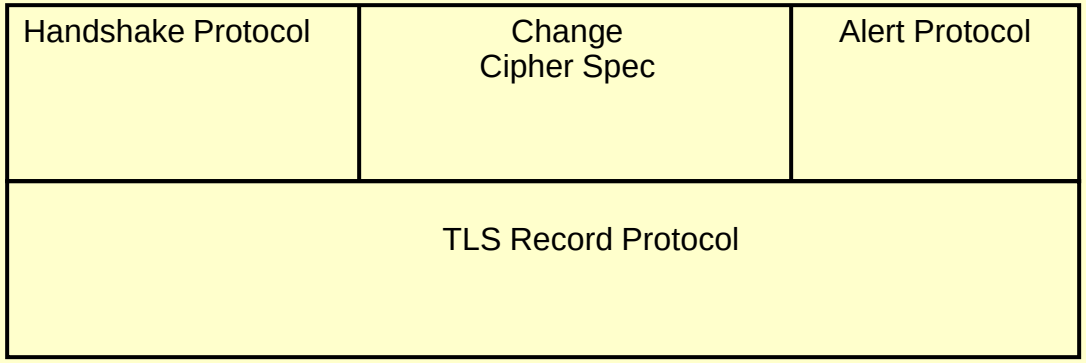
- **Evolved through**
  - **Unreleased v1 (Netscape)**
  - **Flawed-but-useful v2**
  - **Version 3 from scratch**
  - **Standard TLS1.0**
    - » **SSL3.0 with minor tweaks, hence Version field is 3.1**
- **Defined in RFC2246,**
  `http://www.ietf.org/rfc/rfc2246.txt`
- **Open-source implementation at**
  `http://www.openssl.org/`

Overview

- **Establish a session**
  - **Agree on algorithms**
  - **Share secrets**
  - **Perform authentication**
- **Transfer application data**
  - **Ensure privacy and integrity**

Architecture

- **Record Protocol to transfer application and TLS information**
- **A session is established using a Handshake Protocol**



Architecture(1)

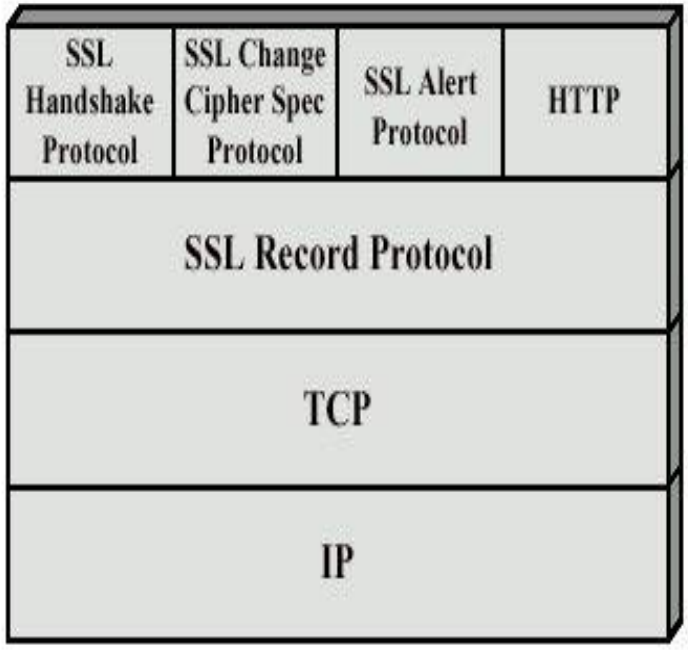


Figure 14.2 SSL Protocol Stack

- **SSL vs. TLS**
  - Layered on top of TCP to provide a reliable end-to-end secure service
  - SSL versions 1.0, 2.0, 3.0, 3.1
  - Netscape protocol
  - Later refitted as IETF standard TLS
  - TLS 1.0 very close to SSLv3.1
- **uses TCP to provide a reliable end-to-end service. SSL is not single protocol but, Two layers of protocol**
  - SSL Record Protocol
  - Three higher layer
    - » SSL Handshake protocol
    - » SSL Change Cipher Spec protocol
    - » SSL Alert protocol

Architecture(2) – Connection, Session

- **Two important SSL concepts**
  - **SSL connection**
    - » **A transport (in the OSI 7 layer) that provides a suitable type of service**
    - » **Peer-to-peer relationships**
    - » **The connections are transient (ie. Temporary)**
    - » **Every connection is associated with one session**
  - **SSL session**
    - » **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**

Architecture(3)-States parameters

- There are actually a number of states associated with each session
- Once session is established, there is a current operating state for both read and write
- In addition, during the Handshake Protocol, pending read and write states are created
- Upon successful conclusion of the Handshake Protocol, the pending states become the current states
- Session state parameters
  - Session Identifier (server to identify an active or resumable session state)
  - Peer Certificate (X509.v3 certificate, may be null)
  - Compression Method - Cipher spec (hash size)
  - Master secret (48 byte secret shared by client and server)
  - Is resumable (flag indicates , session can be used to initiate new conntections)
- Connection state parameters
  - Server and client random (byte sequence chosen by server and client for each conntection)
  - Server write MAC secret (secret key used MAC operations sent by the server)
  - Client write MAC secret(…by client) -
  - Server write key (conventional encry . Key by server and decry. by client)
  - Client write key (encry. Client and decry. Server) - Initialization vectors (IV)
  - Sequence numbers (Each party maintain a sequence number)

SSL Record Protocol(1)

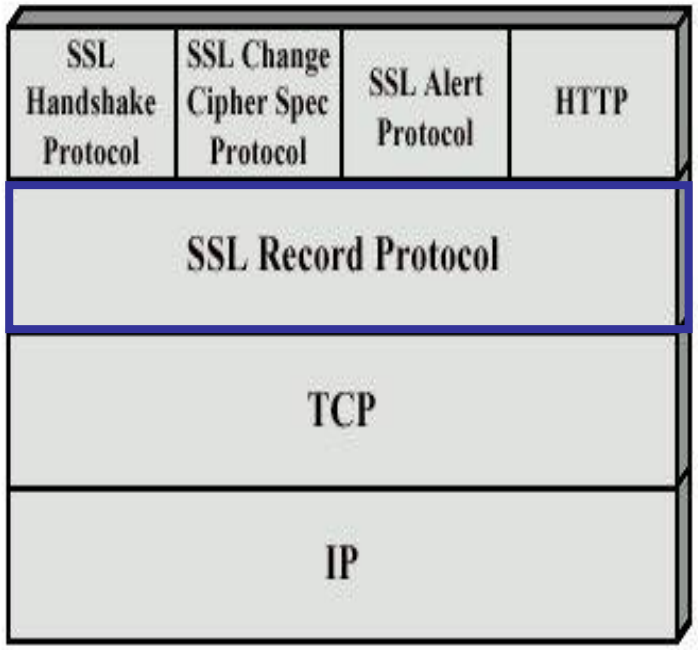


Figure 14.2 SSL Protocol Stack

- **Two services for SSL connections provided by SSL Record protocol**
  - **Confidentiality : conventional encryption of SSL payloads**
  - **Message Integrity : MAC**
  - **Handshake protocol defines both shared secret keys those are used for conventional encryption of SSL payloads and are used to form a message authentication code.**

SSL Record Protocol(2)

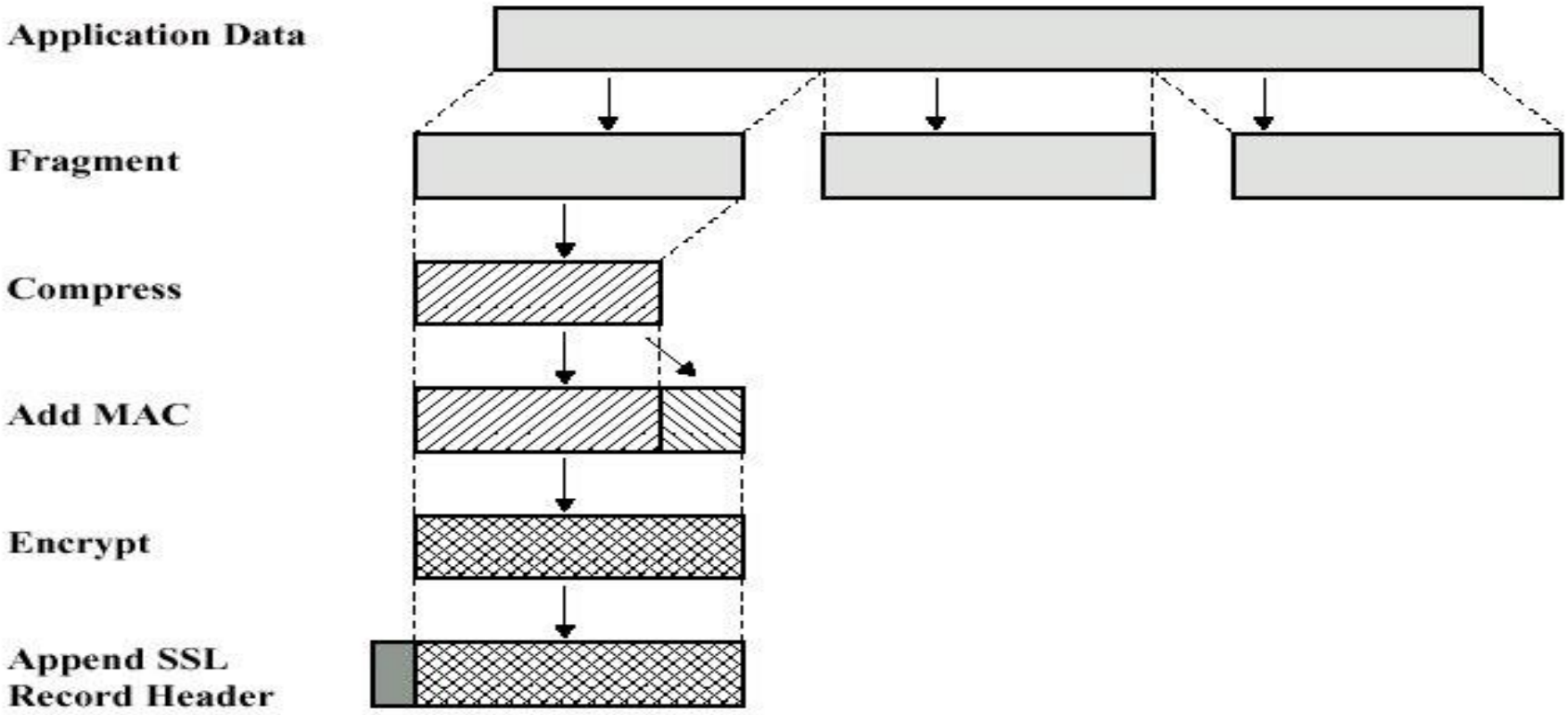


Figure 14.3 SSL Record Protocol Operation

SSL Record Protocol(3)

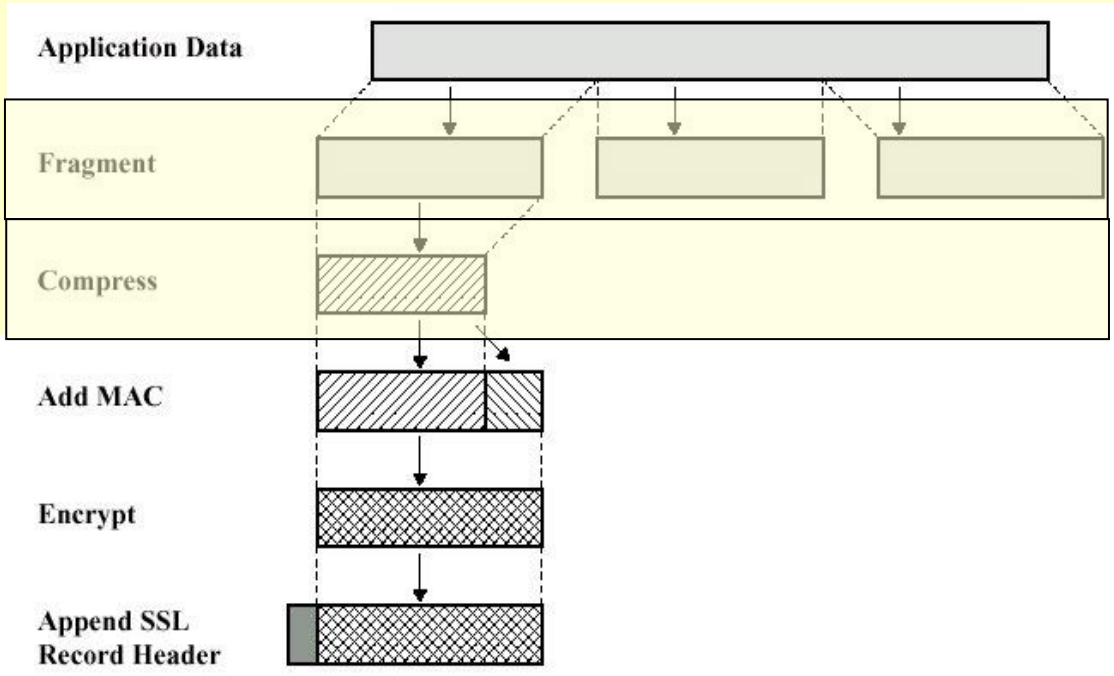


Figure 14.3 SSL Record Protocol Operation

- **1st : Fragmentation**
  - **2^{14} bytes(16384 bytes) or less**
- **2nd : Compression**
  - **Optionally applied**
  - **Must be losses**
  - **May not increase the content length by more than 1024bytes**
  - **In SSLv3, no compression algorithm is specified, so the default compression algorithm is null**

SSL Record Protocol(4)

- **3rd : Add MAC**
  - **A shared secret key used**

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 algo; MD5 or SHA-1
pad_1 = the byte 0x36(0011 0110) repeated 48times(384 bits) for MD5 and 40times(320bits) for SHA-1
pad_2= the byte 0x5c(0101 1100) repeated 48times for MD5 and 40times 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, the plaintext fragment)

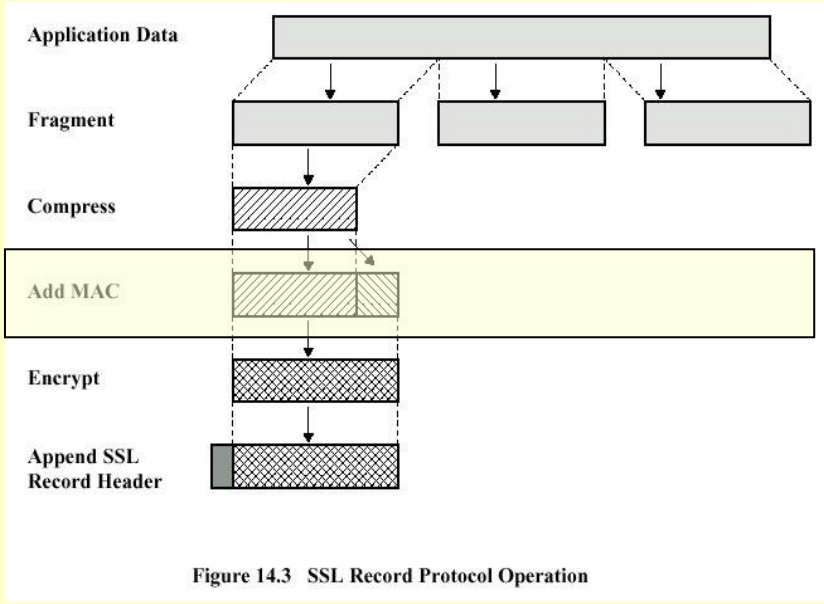


Figure 14.3 SSL Record Protocol Operation

SSL Record Protocol(5)

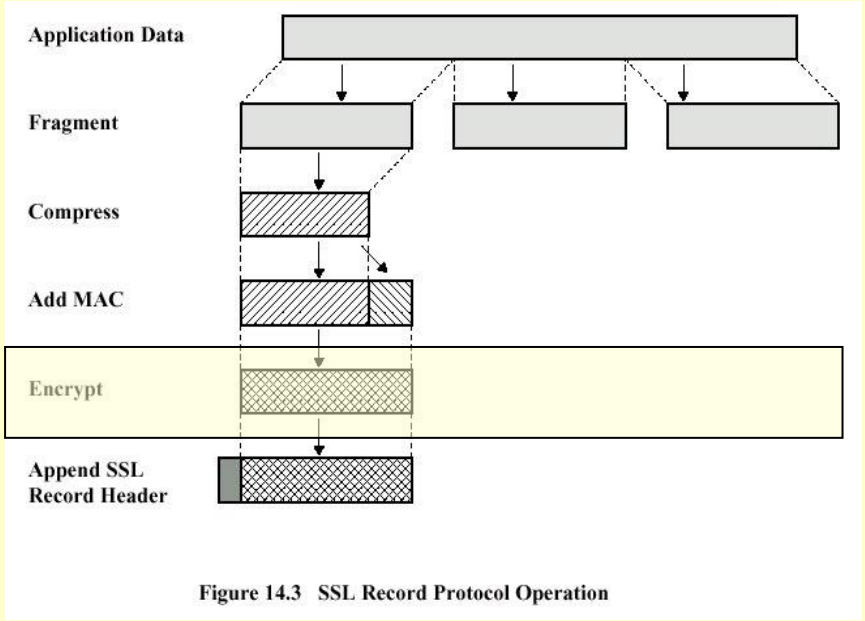


Figure 14.3 SSL Record Protocol Operation

- **4th : Encryption**
  - **Symmetric encryption**
  - **Algorithm used**
    - » **Stream cipher**
      - **The compressed message plus the MAC are encrypted**
      - **RC4-40, RC4-128**
    - » **Block cipher**
      - **Padding may be added**
      - **IDEA, RC2-40, DES-40, DES, 3DES,Fortezza**
      - **The total amount of padding is the smallest amount such that the total size of the data to be encrypted is a multiple of the cipher's block length**

        ex) Plain text : 58 bytes with a MAC of 20 bytes that is encrypted using a block length of 8 bytes = padding.length byte(1) + 1 byte padding

SSL Record Protocol(6)

- **5th : Append SSL record header**
  - **Content type(8bits)**
    - » **change_cipher_spec**
    - » **alert**
    - » **handshake**
    - » **application_data**
  - **Major version(8) – SSLv3**
  - **Minor version(8) - SSLv3 value 0**
  - **Compressed length(16) : length in bytes less than 2^{14} + 2048**

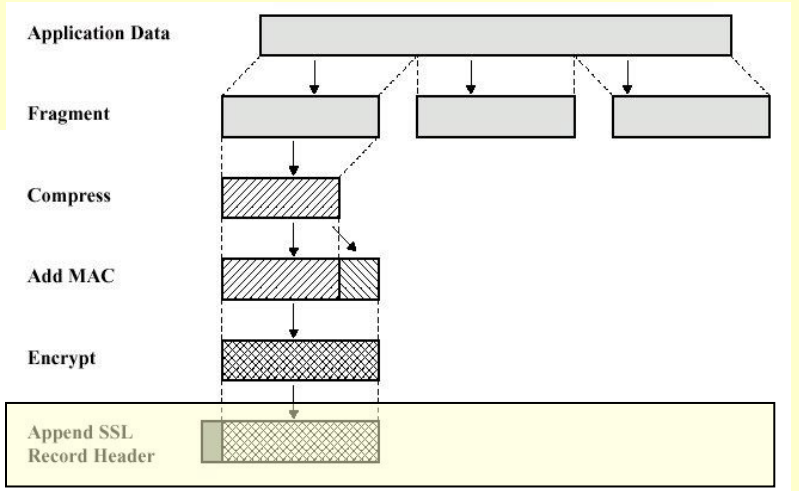


Figure 14.3 SSL Record Protocol Operation

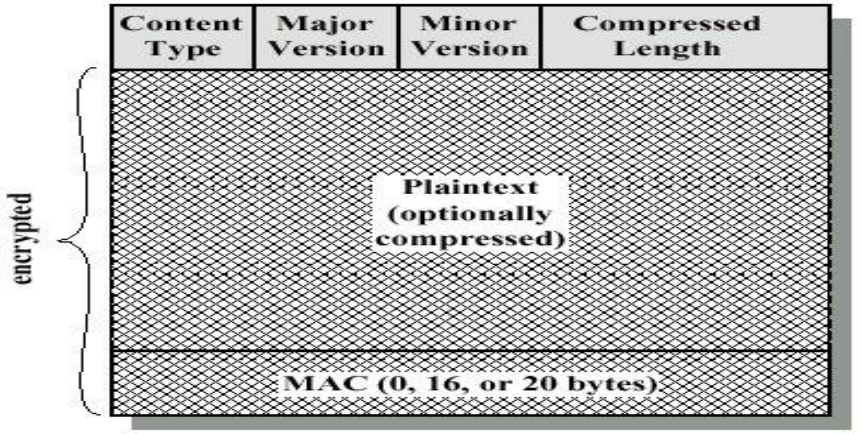


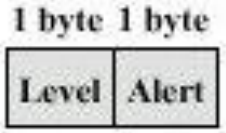
Figure 14.4 SSL Record Format

Change Cipher Spec & Alert Protocol(1)

- **Change Cipher Spec Protocol**
  - Simplest protocol
  - Consists of a single message, which consists of a single byte with the value 1
  - 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**
  - Used to convey SSL-related alerts to the peer entity
  - Consists of two bytes
    - » Level : conveys the severity of the message
      - Warning(1)
      - Fatal(2) – immed. terminates the connection, other connection may continue, but no new connection may be established.
    - » Alert : a code that indicates the specific alert



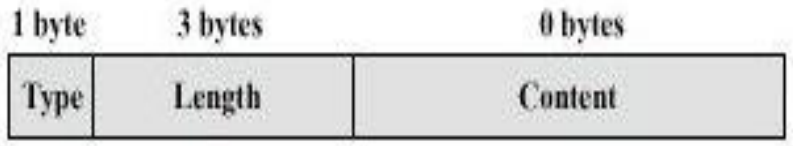
(b) Alert Protocol

Alert Protocol(2)

- **Codes for alerts**
  - **Alerts that are always fatal**
    - **unexpected_message : An in appropriate message was received**
    - **bad_record_mac : An incorrect MAC was received**
    - **decompression_failure : The decompression function received improper input**
    - **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 remainder of alerts**
    - **close_notify :Notifies the recipient that the sender will not send any more messages on this 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**
    - **unsupported_certificate :The type of the received certificate is not supported**
    - **certificate_revoked : revoked by its signer**
    - **certificate_expired : has expired**
    - **certificate_unknown : Some other unspecified issue arose in processing the certificate, rendering it unacceptable**

Handshake Protocol(1)

- The most complex part of SSL
- Allows sever and client
  - To authenticate each other
  - To negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in an SSL record
- Used before any application data is transmitted
- Consists of a series of messages exchanged by client and server



(c) Handshake Protocol

- Three fields
  - Type(1byte) : one of 10 messages
  - Length(3byte)
  - Content(>=1 byte) : parameters associated with this message

Handshake Protocol(2)

Table 14.2 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

Phase 1

Phase 2

Phase 3

Phase 4

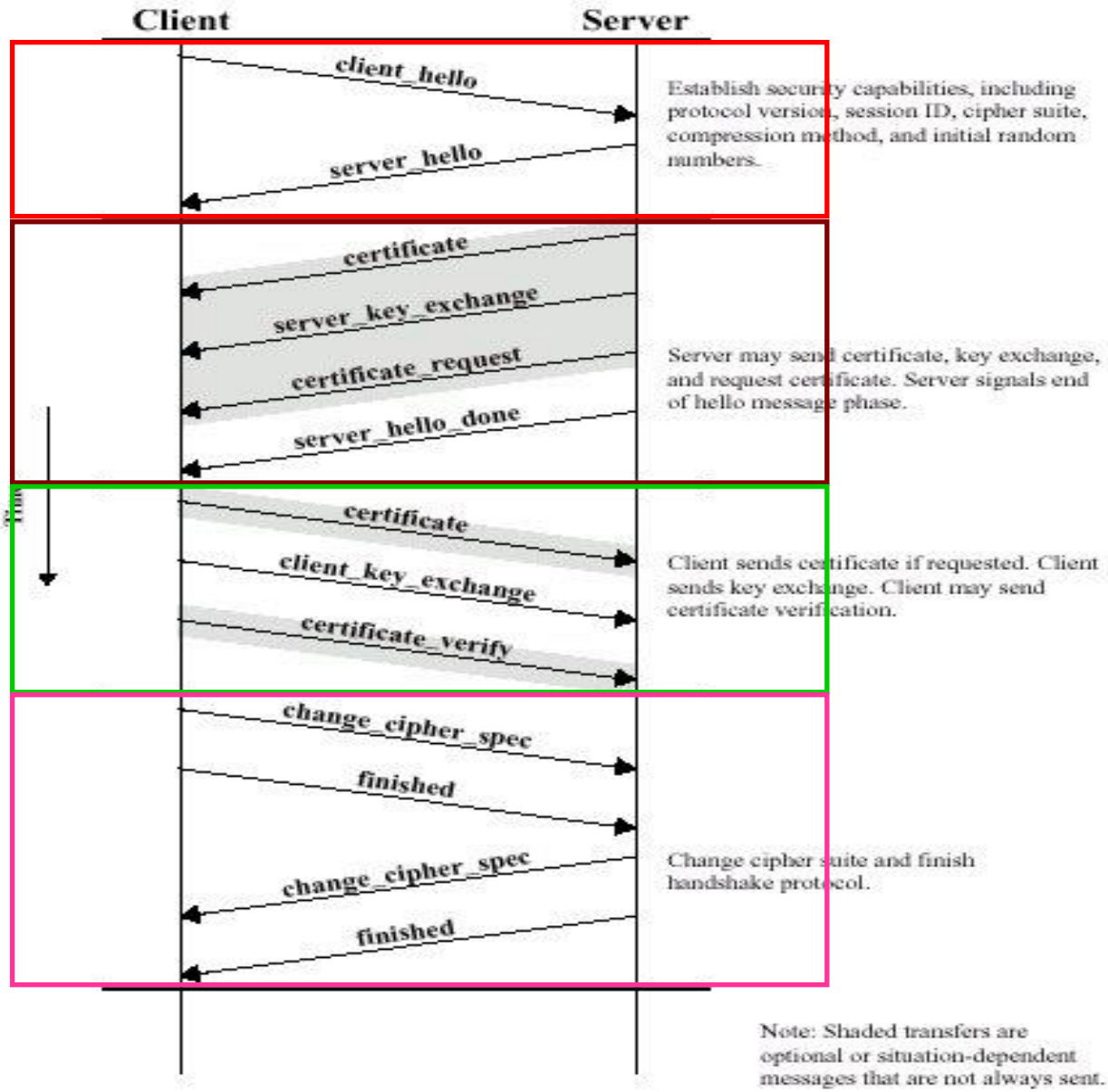


Figure 14.6 Handshake Protocol Action

Handshake Protocol(3)

- **Phase 1 –Establish Security Capabilities**
  - **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**
    - **Version :The highest SSL version**
    - **Random : 32bit timestamp & 28bytes Secure Random Number**
    - **SessionID : Variable-length session identifier**
    - **Cipher Suite : A list that contains the combinations of cryptographic algorithms supported by client, in decreasing order of preference**
    - **Compression Method : The list of compression methods supported by client**
  - **Server sends "server_hello" message with the same parameters**
    - **Version : The lower of the version suggested by the client and the highest supported by the server**
    - **Random : Generated by server**
    - **SessionID : If client is non-zero then,the same with the client. Otherwise, the value for new session**
    - **Cipher Suite : Single cipher suite selected by the server**
    - **Compression Method : The one supported by server**

Handshake Protocol(4)

- **Phase 1 –(continues)**
  - **Cipher Suite**
    - » **Key exchange methods supported**
      - **RSA : The secret key is encrypted with the receiver's public key. A public key certificate for the receiver's key should be available**
      - **Fixed Diffie-Hellman (DH): Server's certificate contains DH public parameters signed by CA. Client provides its DH public key parameters either in certificate or in a key exchange message**
      - **Ephemeral DH : DH public keys are exchanged, signed using sender's private RSA or DSS key. The receiver can use the corresponding public key to verify the signature**
      - **Anonymous DH : The base DH algo is used with no authentication. Vulnerable to man-in-the-middle attack**
      - **Fortezza**
    - » **CipherSpec**
      - **CipherAlgorithm : RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza**
      - **MACAlgorithm : MD5, SHA-1**
      - **CipherType : Stream or block**
      - **IsExportable : True or false**
      - **HashSize : 0, 16(for MD5), or 20(for SHA-1) bytes**
      - **Key Material : A sequence of bytes used in generating the write key**
      - **IV Size : The size of IV for CBC**

Handshake Protocol(5)

- **Phase 2 –Server Authentication and Key Exchange**
  - **The server begins by sending its certificate**
  - **Certificate(X.509) : except anonymous Diffie-Hellman**
  - **Server_key_exchange()**
    - » **Anonymous DH : prime + primitive root**
    - » **Ephemeral DH : prime + primitive root + signature**
    - » **RSA key exchange, in which the server is using RSA but has a signature-only RSA key: temporary RSA public key + signature**
    - » **Fixed DH or RSA key exchange : No need**
    - » **Signature is created by taking the hash of a message and encrypting it with the sender's private key**
      - **hash(ClientHello.random||ServerHello.random||ServerParams)**
  - **Certificate request : Non-anonymous server(server not using anonymous DH) can request certificate from the client**
    - » **Certificate_type : includes public key algorithm and its use**
    - » **Certificate_authorities : a list of the distinguished names of acceptable certificate authorities**
  - **Server_hello_done (no parameter)**

SLO-1 & SLO-2 :
Public Key Infrastructure (PKI)

Overview

- **Introduction**
- **Building Blocks**
- **Certificates**
- **Organization**
- **Conclusions**

Introduction

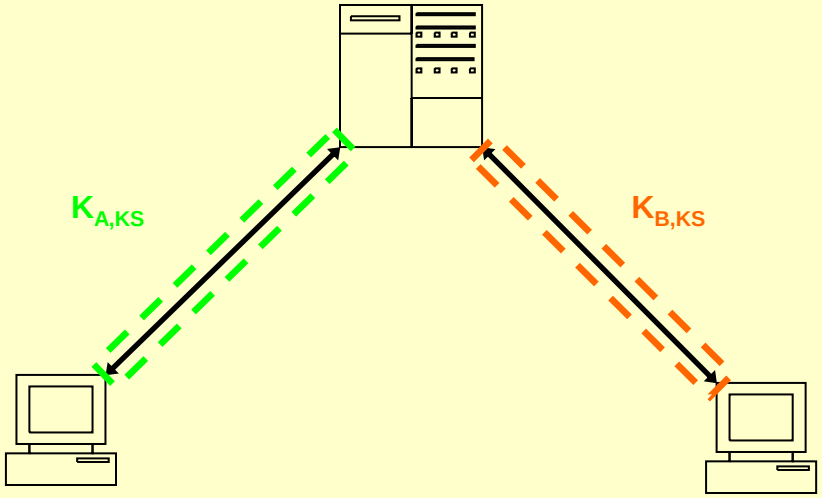
In the beginning there were shared secret keys

- **Early cryptographic systems had to use the same key for encryption and decryption**
- **To establish an encrypted channel both users needed to find out this key in some secure fashion**
  - **Limited – Users could meet and exchange the key**
  - **Flexible – Users could use a key server**

Introduction
Key Exchange – User to User

- **This exchange eliminates a communication channel that could be attacked**
- **Limited - Users must meet all other users**
  - **In a system with n users, number of meetings is on the order of $O(n^2)$**
- **Users must recognize each other or show proper identification**

Introduction
Key Exchange – Key Server



- Each user has set to up a key with the Key Server
- Key Server creates and transmits secure session keys to users
- Flexible – Users need only have a prior established key with the Key Server
  - For a system with n users only (n) meetings must occur
- Key Server takes care of the initial validation of user's identities

Building Blocks

- **Cryptographic tools**
- **Putting them together**
- **Names**
- **Time**
- **A secure communication session**

Building Blocks
Cryptographic Tools

- **Symmetric Key Cryptography**
  - **Encryption: $SE_K(M) = C$**
  - **Decryption: $SD_K(C) = M$**
  - **Secure as long as only communicating users know K**
  - **Having K lets one read C**
  - **Fast to calculate**
- **Public Key Cryptography**
  - **Encryption: $PE_{K+}(M) = C$**
  - **Decryption: $PD_{K-}(C) = M$**
  - **Secure as long K- is only known by the receiver**
  - **Having K- lets one read C, but having K+ does not**
  - **Slow to calculate**

Building Blocks
Cryptographic Tools

- **Digital Signatures**
  - **Sign: $PE_{K-}(H(M)) = S$**
  - **Verify: $PD_{K+}(S) = H(M)$**
  - **Reliable as long as only the signer knows K-**
  - **Having K- allows one to sign, having K+ only allows one to verify the signature**
  - **Slow to calculate**
  - **K's + and - could just be a user's public and private keys**

Building Blocks
Putting Them Together

- **Symmetric cryptography is used for majority of communications**
- **Public Key cryptography is used for exchanging Symmetric keys**
- **Digital Signatures are used to validate Public Keys**

Building Blocks
Names

- **A name in PKI must be unique to a user**
- **Assigning these names presents similar difficulties as found in other areas of Distributed Systems**
- **Without proper and well thought out naming PKI is pretty much useless**

Building Blocks
Time

- **A PKI must know the current time**
- **Much of a PKI's security relies on having an accurate clock**
- **For the most part, time does not need to be known extremely reliably and being off by a minute will usually not be an issue**

Building Blocks
A Secure Communications Session

- **Alice and Bob wish to set up a secure communications channel**
- **They use Public Key Cryptography to exchange a Symmetric key**
  - **Alice: Private PK = $K-_A$, Public PK = $K+_A$**
  - **Bob: Private PK = $K-_B$, Public PK = $K+_B$**
  - **Time T and random Symmetric Key K_S**
  - **Simplified example:**

1: Alice -> Bob: $PE_{K+B}(Alice, T, K+_A, PE_{K-A}(T, K_S))$

2: Bob -> Alice: $PE_{K+A}(T, K_S)$

3: Alice <-> Bob: $SE_{KS}(M_i)$

Certificates

- **What they are**
- **How they are issued**
- **How they are distributed**
- **How they are revoked**

Certificates
What they are

- **The issue with building a secure session is that it assumes that both Alice and Bob know each others public keys**
- **We need some way for them to learn this besides meeting each other (otherwise we are in the same predicament as with Symmetric Key exchange meetings)**
- **We could use a similar strategy to the Key Server but can we do better?**

This is where Certificates come in…

Certificates
What they are

- **A Certificate is a combination of a user's public key, unique name, Certificate start and expiration dates, and possibly other information**
- **This Certificate is then digitally signed, by some Trusted 3rd Party, with the signature being attached to the rest of the Certificate**
- **This Signed Certificate is commonly referred to as just the user's Certificate**
- **The Certificate for a user Bob, signed by signer Tim, in essence states**

  "I Tim certify that this Public Key belongs to Bob"

Certificates
How they are issued

- **The users of a PKI must place their trust in a 3rd Party to carefully verify a user's identity before signing his or her public key**
- **Each user generates their own Public-Private Key pair and Certificate**
- **A user then verifies them self to the 3rd Party and shows his or her Certificate's content. At this point the third party will sign the Certificate.**

Certificates
How they are distributed

- **Users are free to distribute their signed Certificates over any medium, public or private, without concern**
- **Other users may acquire this Certificate from any source and check the 3rd Party's signature for tampering**
- **If the signature is good then the other users know that the 3rd Party affirms that the Certificate belongs to the user who is listed in the Certificate**

Certificates
How they are Revoked

- **Periodically Certificates may become compromised, requiring a Certificate Revocation**
- **A Certificate Revocation message is simply a message signed by $K-_i$ (the private version of the Certificate's $K+_i$) saying that the Certificate is revoked**
- **A PKI will have a database of revoked Certificates (a Certificate Revocation List, CRL) that users may access periodically for the latest list of revoked Certificates**
- **An alternative to certificate revoking is to set the expiration time to very shortly after the issue time. Thus every key in this system is revoked so rapidly that we do not need to worry what may happen to the compromised key**

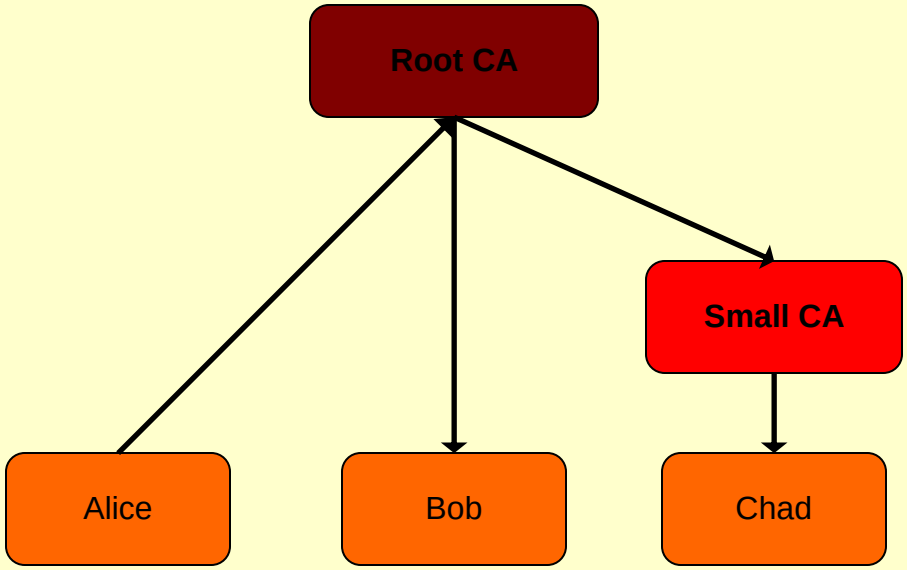
Organization

- **What is “Trust”?**
- **How do we organize a PKI to disseminate trust?**

Organization
Trust

- **Trust is based on real world contractual obligations between a 3rd Party and users [2]**
- **This Trusted 3rd Party is referred to as a Certificate Authority (CA)**
- **In other models trust is based on personal relationships that don't have a contractual basis (e.g. PGP)**
- **Users may allow a CA to delegate their trust**
- **This delegation of trust is what allows us to build large PKI's**

Organization
Trust

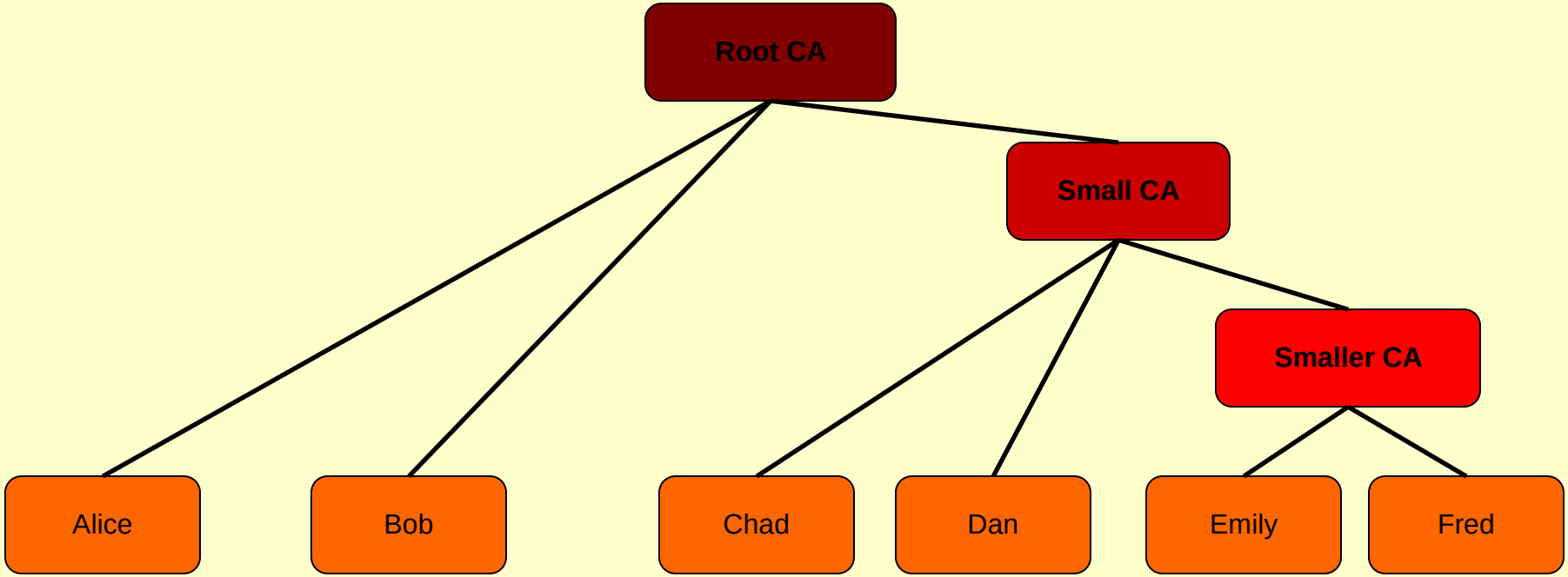


- **If Alice trusts Root CA then she trusts Bob's Certificate signed by Root CA**
- **If Alice trusts Root CA to delegate her trust to others then she trusts Chad's Certificate signed by Small CA**

Organization
Organizing a PKI

- **A PKI may be organized based on a variety of models using delegation of trust**
  - **Strict Hierarchy**
  - **Networked**
  - **Web Browser**
  - **PGP**

Organization
Strict Hierarchy

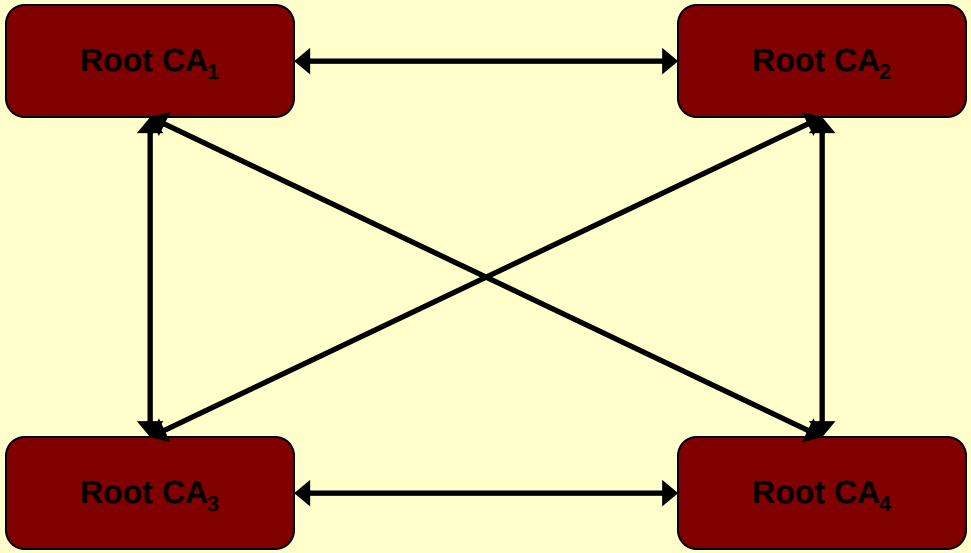


- **All users trust Root CA**
- **Root CA may delegate that trust to other CA's who in turn may be allowed to delegate that trust**
- **In this way a PKI may grow without all the burden being placed on Root CA**

Organization
Networked

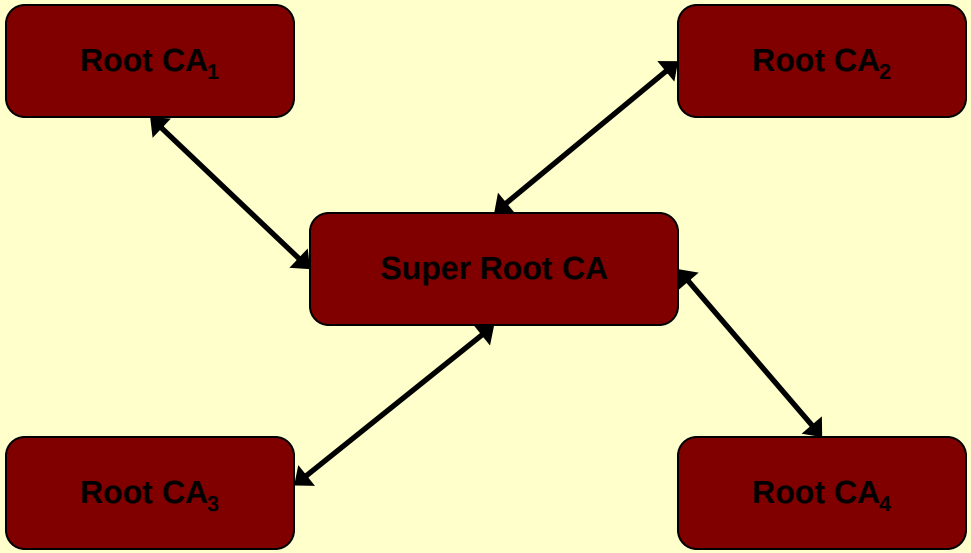
- The Networked model addresses what to do when two or more PKIs wish to join together or merge
- Two techniques
  - Mesh
  - Hub-and-Spoke
- We only need the Root CAs of each PKI to participate in this model

Organization
Networked – Mesh



- **Every Root CA signs every other Root CA's Certificate**
- **Hard to join a large numbers of CAs**

Organization
Networked – Hub-and-Spoke



- The Root CAs come together to create the Super Root CA
- Each Root CA signs the Super Root CA's certificate while the Super Root CA signs each of theirs
- Easier to join large numbers of CAs
- Question becomes, Who gets to manage the Super Root CA?

Conclusions

- A PKI allows us to take the concept of a Key Server and apply it to Public Keys
- It allows greater flexibility then a Key Server in that users do not need to communicate with the Root CA every time a Session Key is needed
- There are a vast variety of models for disseminating trust in a PKI
- Even though PKIs look like an amazing idea, in practice there are numerous problems implementing them on a large scale
  - Who does everyone trust?
  - What format do people use?
  - Security of the multitude of programs that rely on PKIs

Secure Electronic Transaction

- **SET**
  - Open encryption specification, credit card transactions
  - SETv1
    - » by MasterCard and Visa in February 1996.
  - A wide range of companies were involved
    - » IBM, Microsoft, Netscape, RSA, Terisa, and Verisign ….
  - First products are available in 1998.
  - is not a payment system.
  - is security protocols, formats.
- **SET provides**
  - A secure communication channel
  - Trust by the use of X.509v3 digital certificates
  - Ensures privacy
- **SET is defined in**
  - Book1 : Business Description (80 pages)
  - Book2 : Programmer's Guide (629 pages)
  - Book3 : Formal Protocol Definition (262 pages)

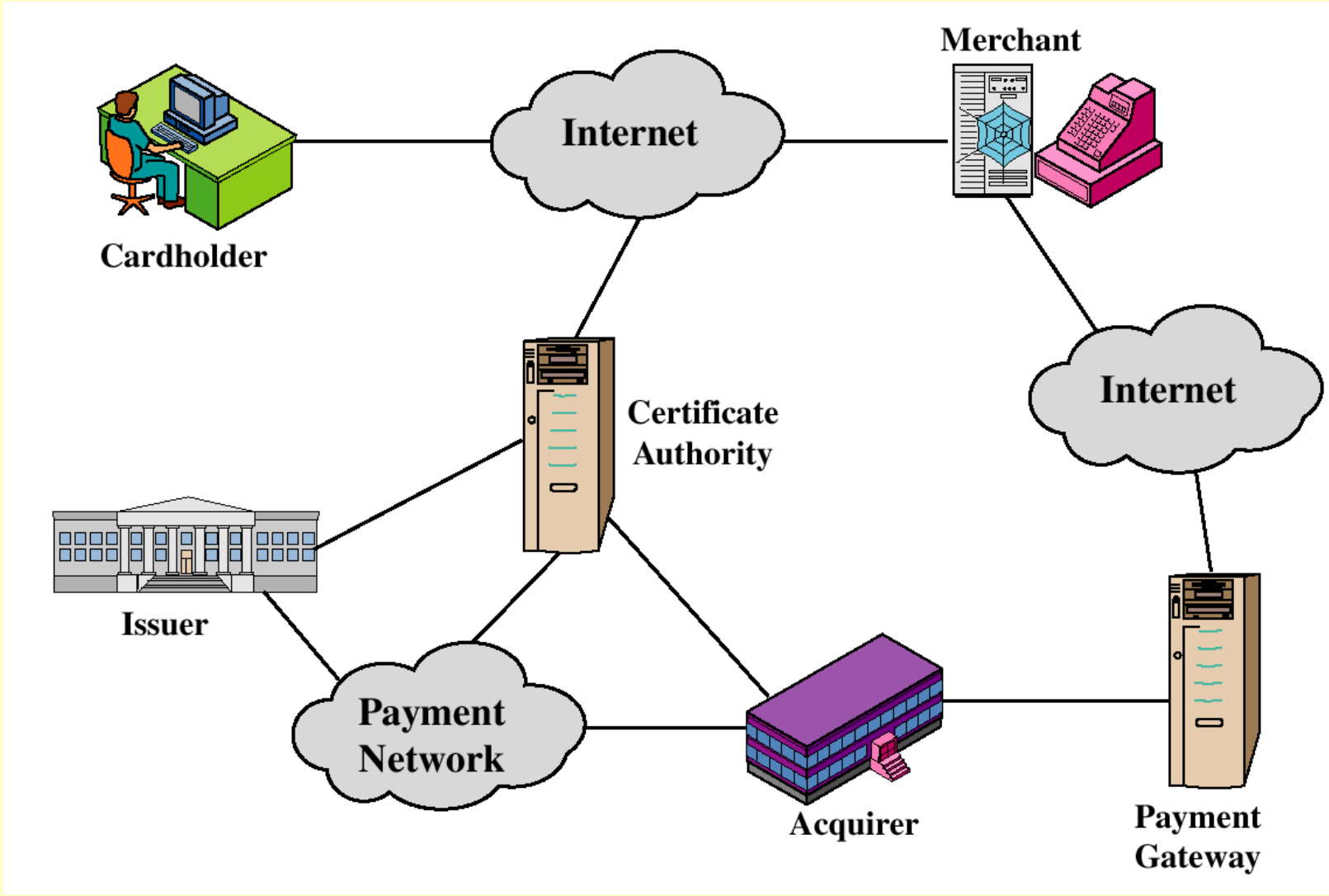
Business Requirements for SET

- **Provide confidentiality of payment and ordering information.**
- **Ensure the integrity of all transmitted data.**
- **Provide authentication**
  - **a cardholder is a legitimate user of a credit card account.**
  - **a merchant can accept credit card transactions.**
- **Ensure the use of the best security practices and system design**
- **Create a protocol that neither depends on transport security mechanisms nor prevents their use.**
- **Facilitate and encourage interoperability among software and network providers.**

Key Features of SET

- **Confidentiality of information**
  - Cardholder account and payment information is secured.
  - Merchant can not learn the cardholder's credit card number.
  - Using DES for confidentiality.
- **Integrity of data**
  - Type of data : Order information, personal data, payment instructions.
  - RSA digital signatures using SHA-1 or HMAC using SHA-1.
- **Cardholder account and Merchant authentication**
  - X.509v3 digital certificates with RSA signatures
- **Interoperability**
  - SET uses specific protocols and message formats.

SET Participants(1)



SET Participants(2)

- **Cardholder**
  - An authorized holder of a payment card (e.g., MasterCard, Visa).
- **Merchant**
  - A person or organization that has goods or services to sell.
- **Issuer**
  - provides the cardholder with the payment card.
  - is responsible for the payment of the debt of the cardholder.
- **Acquirer**
  - establishes an account with a merchant
  - Processing payment card authorizations and payments.
- **Payment Gateway**
  - Processing merchant payment messages.
  - Interfaces between SET and bankcard payment networks.
- **Certification Authority (CA)**
  - Issue X.509v3 public-key certificates.
  - The success of SET will depend on the existence of a CA infrastructure available.

Sequence of events for transaction

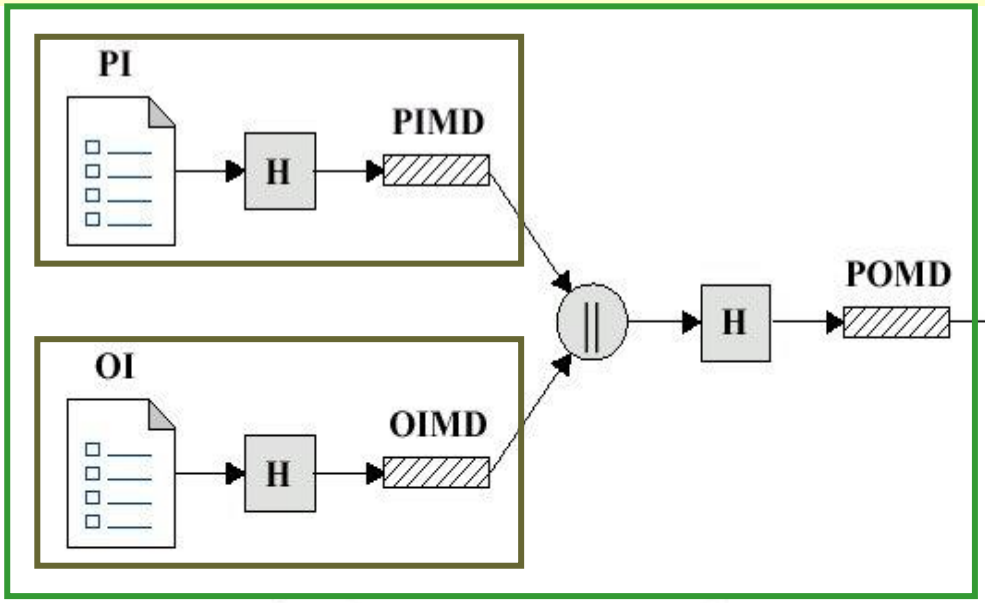
1. The customer opens an account.
2. The customer receives a certificate.
3. Merchants have their own certificates.
   - Two certificates : One for signing message, One for key exchange
4. The customer places an order.
   - Merchant returns an order form containing the list of items, their price, a total price, and an order number.
5. The merchant is verified.
   - Customer verifies that a store is valid or not.
6. The order and payment are sent.
7. The merchant requests payment authorization.
   - Merchant sends the payment information to the payment gateway, requesting authorization.
8. The merchant confirms the order.
9. The merchant provides the goods or service.
10. The merchant requests payment.

Dual Signature(1)

- **Dual Signature**
  - **An Important innovation introduced in SET.**
  - **Link two messages that are intended for two different recipients.**
  - **Customer -> Merchant : Order Information (OI)**
    - » **Merchant does not need to know the details of the customer's credit card number**
  - **Customer -> Bank : Payment Information (PI)**
    - » **the bank does not need to know the details of the customer's order**
- **The customer is afforded extra protection in terms of privacy by keeping these two items separate.**
- **The two items must be linked.**
  - **order information and payment information.**

Dual Signature(2)

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



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_c = Customer's private signature key

Figure 14.9 Construction of Dual Signature

Dual Signature(3)

- **Customer makes dual signature.**
  - $DS = E_{KRc}[H(H(PI)||H(OI))]$
- **Merchant verifies the signature.**
  - **Use DS, OI, PIMD, and customer's public key**
  - **Merchant computes and Compares two quantities;**
    - » $H(PIMD||H(OI))$ **and** $DS = D_{KUc}[DS]$
- **Bank verifies the signature.**
  - **Use DS, OIMD, PI, and customer's public key**
  - **Merchant computes and Compares two quantities;**
    - » $H(H(PI)||H(OIMD))$ **and** $DS = D_{KUc}[DS]$

※ Customer has linked the OI and PI and can prove the linkage.

SET Transaction Types(1)

- **Cardholder registration**
  - **Cardholders must register with a CA before they can send SET messages to merchants.**
- **Merchant registration**
  - **Merchants must register with a CA before they can exchange SET messages with customers and payment gateways.**
- **Purchase request**
  - **Message from customer to merchant containing OI for merchant and PI for bank.**
- **Payment authorization**
  - **Exchange between merchant and payment gateway to authorize a given amount for a purchase on a given credit card account.**
- **Payment capture**
  - **Allows the merchant to request payment from the payment gateway.**

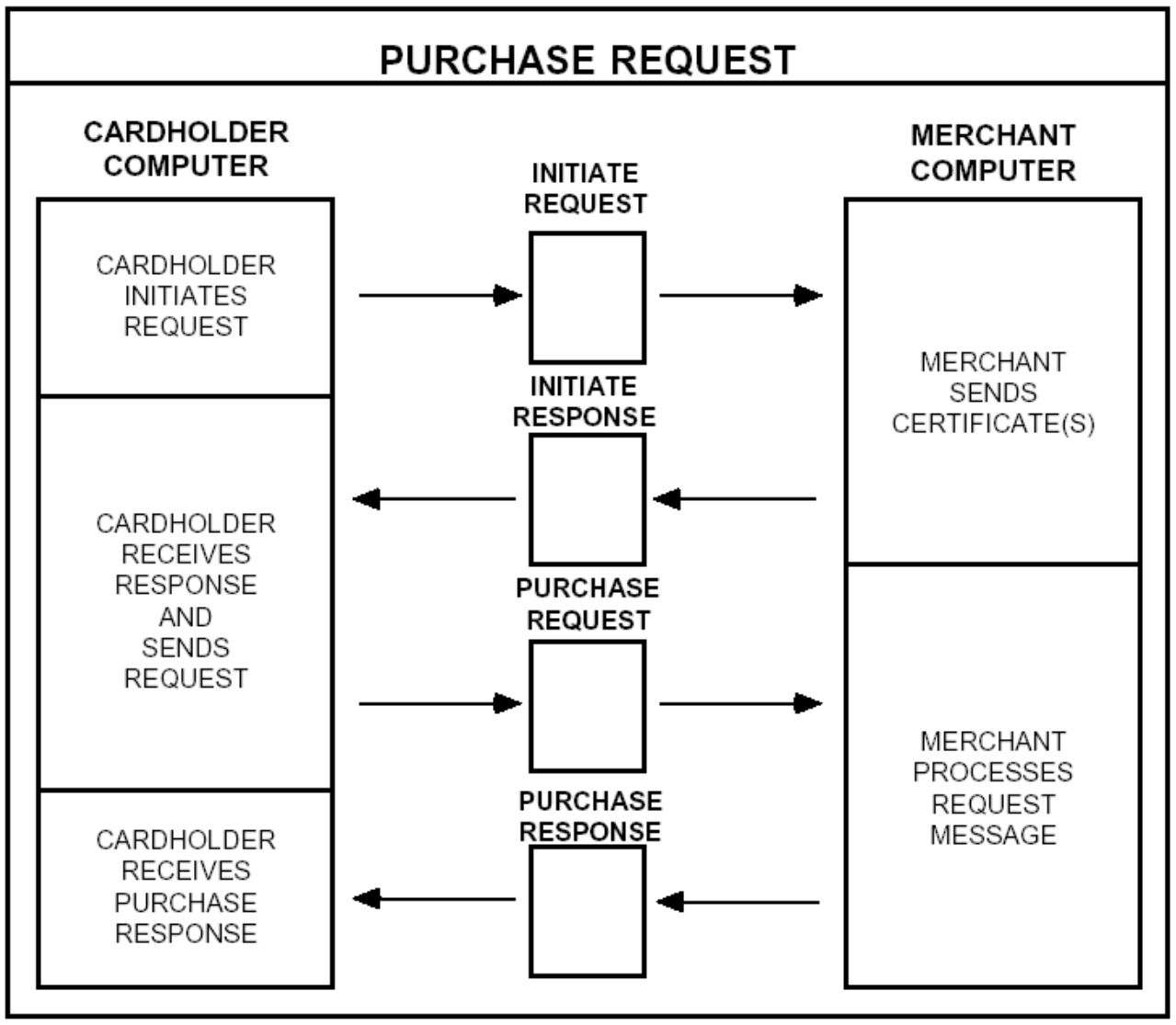
SET Transaction Types(2)

Certificate inquiry and status	If the CA is unable to complete the processing of a certificate request quickly, it will send a reply to the cardholder or merchant indicating that the requester should check back later. The cardholder or merchant sends the *Certificate Inquiry* message to determine the status of the certificate request and to receive the certificate if the request has been approved.
Purchase inquiry	Allows the cardholder to check the status of the processing of an order after the purchase response has been received. Note that this message does not include information such as the status of back ordered goods, but does indicate the status of authorization, capture and credit processing.
Authorization reversal	Allows a merchant to correct previous authorization requests. If the order will not be completed, the merchant reverses the entire authorization. If part of the order will not be completed (such as when goods are back ordered), the merchant reverses part of the amount of the authorization.
Capture reversal	Allows a merchant to correct errors in capture requests such as transaction amounts that were entered incorrectly by a clerk.

SET Transaction Types(3)

Credit	Allows a merchant to issue a credit to a cardholder's account such as when goods are returned or were damaged during shipping. Note that the SET *Credit* message is always initiated by the merchant, not the cardholder. All communications between the cardholder and merchant that result in a credit being processed happen outside of SET.
Credit reversal	Allows a merchant to correct a previously request credit.
Payment gateway certificate request	Allows a merchant to query the Payment Gateway and receive a copy of the gateway's current key-exchange and signature certificates.
Batch administration	Allows a merchant to communicate information to the Payment Gateway regarding merchant batches.
Error message	Indicates that a responder rejects a message because it fails format or content verification tests.

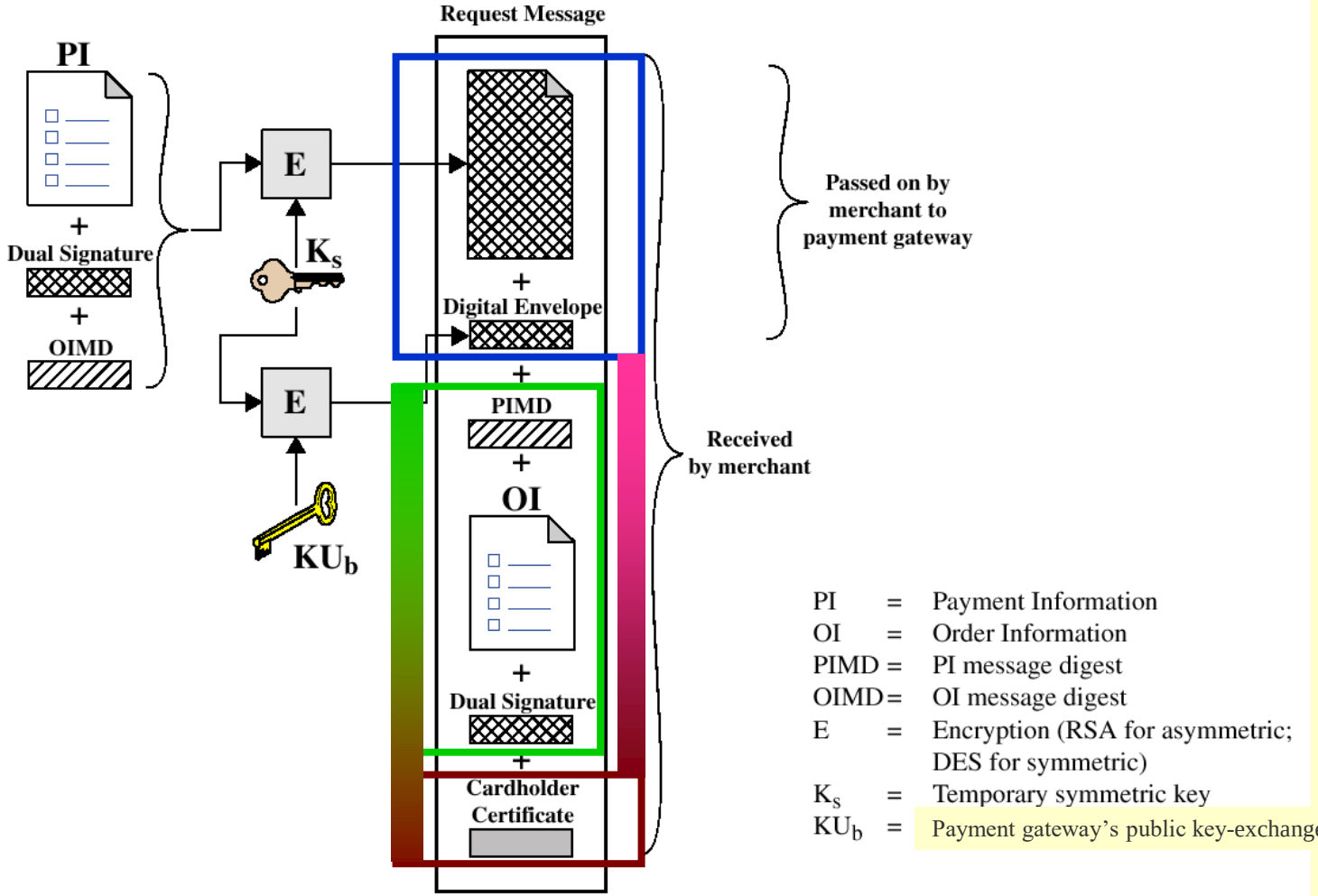
Purchase Request(1)



Purchase Request(2)

- **All of the preceding occurs without the use of SET.**
  - **Customer : visit the cyber store, browsing, selecting, and ordering.**
  - **Merchant : sending order form to customer**
- **Purchase request exchange consists of 4 messages.**
  - **Initiate Request, Initiate Response, Purchase Request, and Purchase Response**
- **Initiate Request message**
  - **The customer requests the certificates of the merchant and the payment gateway.**
  - **Brand of the credit card that the customer is using.**
  - **ID of customer and Nonce.**
- **Initiate Response message**
  - **Merchant signs message with his private signature key.**
  - **Nonce from the customer and another Nonce by merchant.**
  - **Transaction ID for this purchase transaction.**
  - **Merchant's signature certificate and Payment gateway's key exchange certificate.**

Purchase Request message(1)

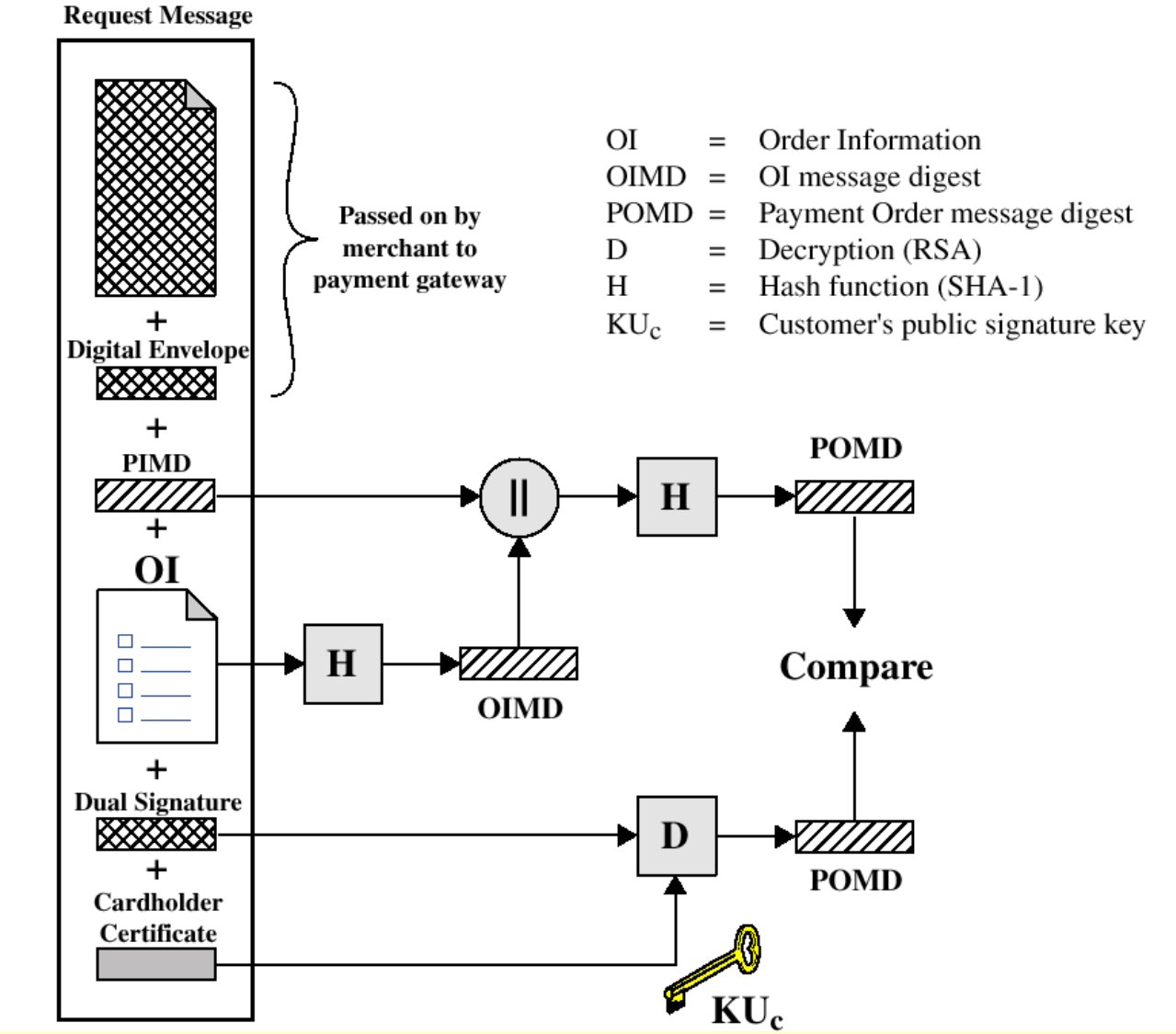


Cardholder sends Purchase Request.

Purchase Request message(2)

- **Purchase-related information**
  - **The merchant sends this information to the payment gateway.**
  - **The PI**
  - **The dual signature signed with the customer's private signature key.**
  - **The OI message digest (OIMD)**
    - » **The OIMD is needed for the payment gateway to verify the dual signature.**
  - **The digital envelope**
    - » **This is formed by encrypting Ks with the payment gateway's public key exchange key.**
- **Order-related information**
  - **This information is needed by the merchant.**
  - **The OI**
  - **The dual signature**
  - **The PI message digest (PIMD)**
- **Cardholder's certificate**

Purchase Request message(3)

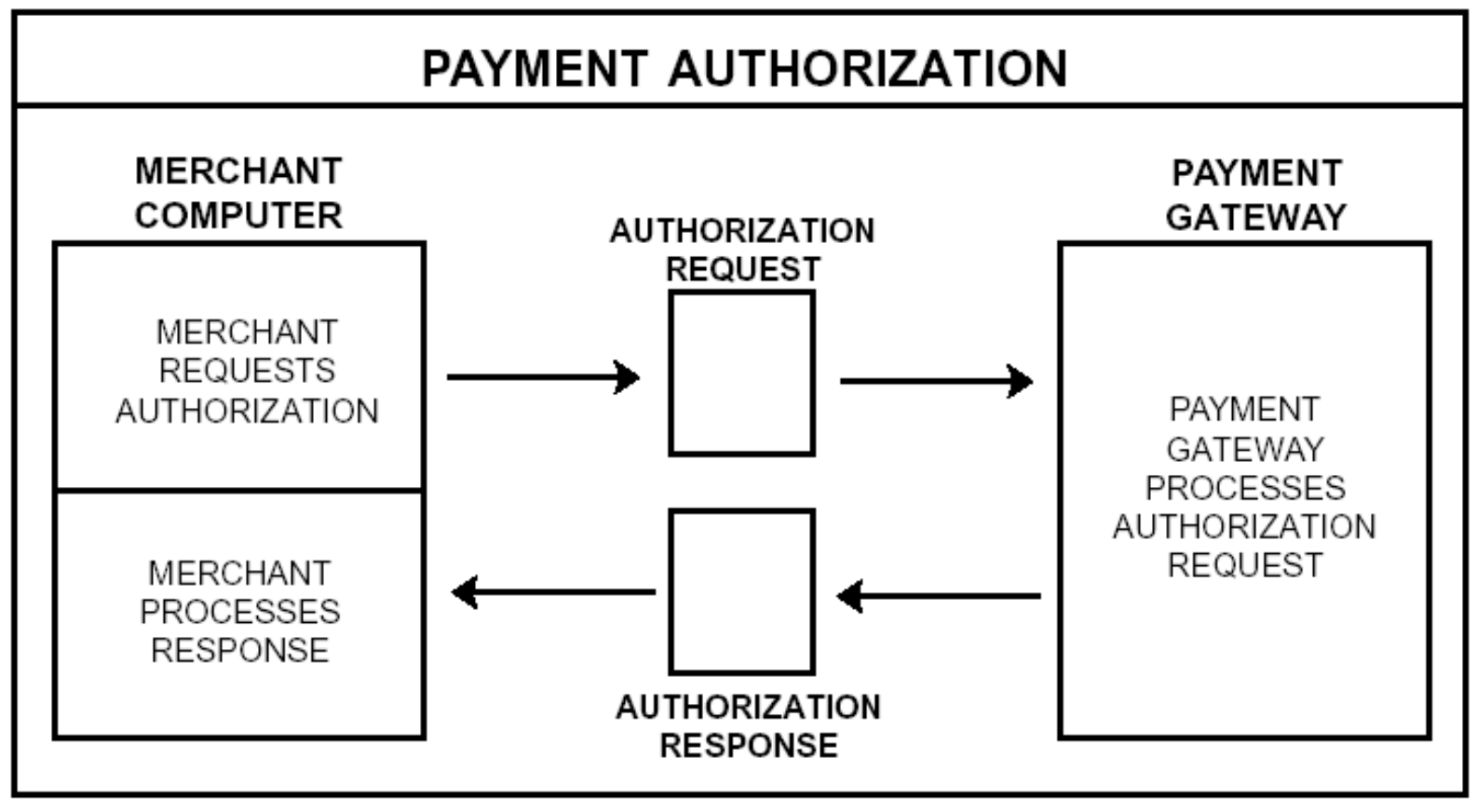


Merchant Verifies Customer Purchase Request.

Purchase Request message(4)

- **Merchant processes purchase request message.**
  - **Verifies the cardholder certificates by means of its CA signatures.**
  - **Verifies the dual signature using the customer's public signature key.**
  - **Processes the order and forwards the payment information to the payment gateway for authorization.**
  - **Sends a purchase response message to the cardholder.**
- **Purchase response message**
  - **It contains that response block and merchant's signature certificate.**
  - **Response block is signed by the merchant using its private signature key.**
- **Cardholder processes purchase response message.**
  - **Verifies the merchant's certificate and then verifies the signature on the response block.**

Payment Authorization



Authorization Request message

- **Purchase-related information**
  - **The PI**
  - **The dual signature**
  - **The OI message digest (OIMD)**
  - **The digital envelope**
- **Authorization-related information**
  - **An authorization block that includes the transaction ID**
  - **A digital envelope**
- **Certificates.**
  - **The cardholder's signature key certificate : used to verify the dual signature.**
  - **The merchant's signature key certificate : used to verify the merchant's signature in authorization block.**
  - **The merchant's key-exchange certificate : needed in the payment gateway's response.**

Actions of the Payment Gateway
(after receiving authorization request message)

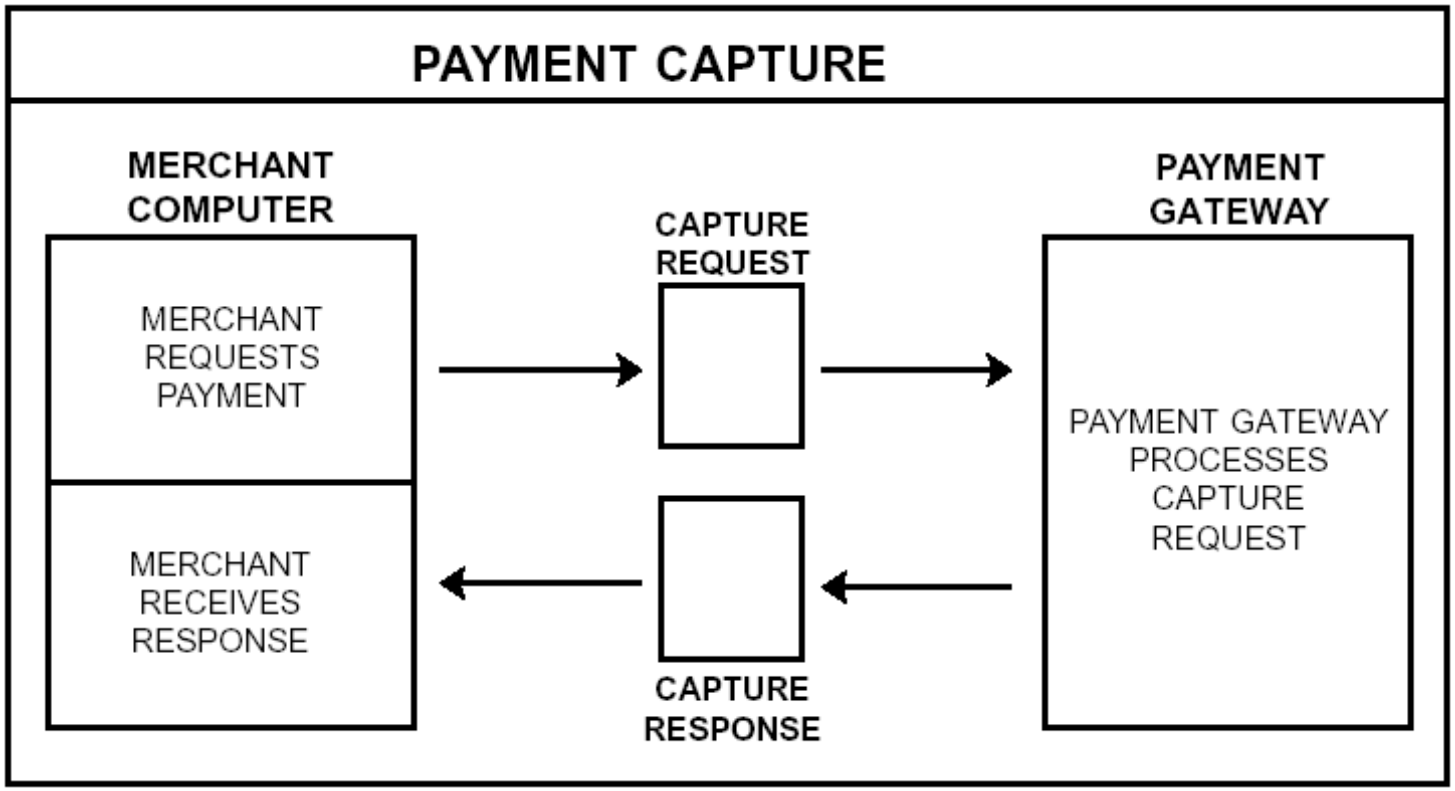
1. Verifies all certificates.
2. Decrypts the digital envelope of the authorization block to obtain the symmetric key and then decrypts the authorization block.
3. Verifies the merchant's signature on the authorization block.
4. Decrypts the digital envelope of the payment block to obtain the symmetric key and then decrypts the payment block.
5. Verifies the dual signature on the payment block.
6. Verifies that the transaction ID received from the merchant matches that in the PI received (indirectly) from the customer.
7. Requests and receives an authorization from the issuer.

Authorization Response message

- **Authorization-related information**
  - **An authorization block, signed with the payment gateway's private signature key and encrypted with a one-time symmetric key generated by the payment gateway.**
  - **A digital envelope that contains the one-time symmetric key encrypted with the merchant's public key-exchange key.**
- **Capture token information**
  - **A capture token, signed with the payment gateway's private key and encrypted with a newly generated one-time symmetric key.**
  - **A digital envelope that contains this one-time symmetric key and cardholder account information encrypted with the payment gateway's public key-exchange key.**
- **Certificate**
  - **The payment gateway's signature key certificate.**

※ With the authorization from the gateway, the merchant can provide the goods or services to the customer.

Payment Capture(1)



Payment Capture(2)

- **Capture Request message**
  - **Capture request block, signed and encrypted.**
    - » **Payment amount, transaction ID**
  - **The encrypted capture token received in the authorization response for this transaction.**
  - **The digital envelope**
  - **Certificates**
- **Capture Response message**
  - **Capture response block, signed and encrypted.**
  - **The digital envelope**
  - **Certificates**

※ The merchant stores the capture response to be used for reconciliation with payment received from the acquirer.