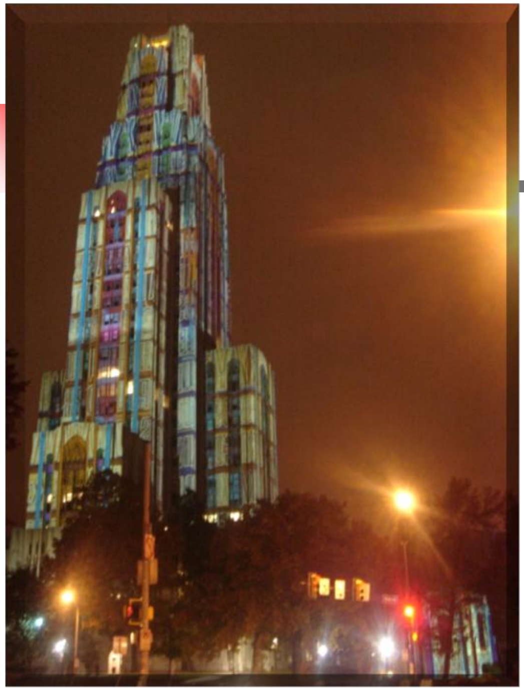
IS 2150 / TEL 2810
Information Security & Privacy



James Joshi
Associate Professor, SIS

Lecture 7
Feb 20, 2013

Key Management
Network Security

Objectives

- Understand/explain the issues related to, and utilize the techniques
  - Key management
    - Authentication and distribution of keys
      - Session key, Key exchange protocols
    - Mechanisms to bind an identity to a key
    - Generation, maintenance and revoking of keys
  - Security at different levels of OSI model
      - Privacy Enhanced email
      - IPSec

Notation

- $X \rightarrow Y : \{ Z \,||\, W \}\, k_{X,Y}$
  - X sends Y the message produced by concatenating Z and W enciphered by key $k_{X,Y}$, which is shared by users X and Y
- $A \rightarrow T : \{ Z \}\, k_A \,||\, \{ W \}\, k_{A,T}$
  - A sends T a message consisting of the concatenation of Z enciphered using k_A, A's key, and W enciphered using $k_{A,T}$, the key shared by A and T
- r_1, r_2 nonces (nonrepeating random numbers)

Interchange vs Session Keys

- Interchange Key
  - Tied to the principal of communication
- Session key
  - Tied to communication itself
- Example
  - Alice generates a random cryptographic key k_s and uses it to encipher m
  - She enciphers k_s with Bob's public key k_B
  - Alice sends $\{ m \} k_s \{ k_s \} k_B$
    - Which one is session/interchange key?

Benefits using session key

- In terms of Traffic-analysis by an attacker?
- Replay attack possible?
- Prevents some *forward search attack*
  - Example: Alice will send Bob message that is either "BUY" or "SELL".
  - Eve computes possible ciphertexts {"BUY"} k_B and {"SELL"} k_B.
  - Eve intercepts enciphered message, compares, and gets plaintext at once

Key Exchange Algorithms

- Goal:
  - Alice, Bob to establish a shared key
- Criteria
  - Key cannot be sent in clear
  - Alice, Bob may trust a third party
  - All cryptosystems, protocols assumed to be publicly known

Classical Key Exchange

- How do Alice, Bob begin?
  - Alice can't send it to Bob in the clear!
- Assume trusted third party, Cathy
  - Alice and Cathy share secret key k_A
  - Bob and Cathy share secret key k_B
- Use this to exchange shared key k_s

Simple Key Exchange Protocol

$$\text{Alice} \xrightarrow{\{\text{ request for session key to Bob }\}\ k_A} \text{Cathy}$$

$$\text{Alice} \xleftarrow{\{\ k_s\ \}k_A\ ,\ \{\ k_s\ \}k_B} \text{Cathy}$$

$$\text{Alice} \xrightarrow{\{\ k_s\ \}\ k_B} \text{Bob}$$

$$\text{Alice} \xleftrightarrow{\{m\}k_s} \text{Bob}$$

What can an attacker, Eve, do to subvert it?

Needham-Schroeder

$$\text{Alice} \xrightarrow{\ \text{Alice} \,\|\, \text{Bob} \,\|\, r_1\ } \text{Cathy}$$

$$\text{Alice} \xleftarrow{\ \{\, \text{Alice} \,\|\, \text{Bob} \,\|\, r_1 \,\|\, k_s \,\|\, \{\, \text{Alice} \,\|\, k_s \,\}\, k_B \,\}\, k_A\ } \text{Cathy}$$

$$\text{Alice} \xrightarrow{\ \{\, \text{Alice} \,\|\, k_s \,\}\, k_B\ } \text{Bob}$$

$$\text{Alice} \xleftarrow{\ \{\, r_2 \,\}\, k_s\ } \text{Bob}$$

$$\text{Alice} \xrightarrow{\ \{\, r_2 - 1 \,\}\, k_s\ } \text{Bob}$$

Questions

- How can Alice and Bob be sure they are talking to each other?
- Is the previous attack possible?
- Key assumption of Needham-Schroeder
  - All keys are secret;
  - What if we remove that assumption?

Needham-Schroeder with Denning-Sacco Modification

$$\text{Alice} \xrightarrow{\text{Alice} \,\|\, \text{Bob} \,\|\, r_1} \text{Cathy}$$

$$\text{Alice} \xleftarrow{\{\, \text{Alice} \,\|\, \text{Bob} \,\|\, r_1 \,\|\, k_s \,\|\, \{\, \text{Alice} \,\|\, T \,\|\, k_s \,\}\, k_B \,\}\, k_A} \text{Cathy}$$

$$\text{Alice} \xrightarrow{\{\, \text{Alice} \,\|\, T \,\|\, k_s \,\}\, k_B} \text{Bob}$$

$$\text{Alice} \xleftarrow{\{\, r_2 \,\}\, k_s} \text{Bob}$$

$$\text{Alice} \xrightarrow{\{\, r_2 - 1 \,\}\, k_s} \text{Bob}$$

Use time stamp *T* to detect replay!

Synchronized Clocks needed!

Otway-Rees Protocol

Alice $\xrightarrow{n \,\|\, \text{Alice} \,\|\, \text{Bob} \,\|\, \{ r_1 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob} \} k_A}$ Bob

Cathy $\xleftarrow[\{ r_2 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob} \} k_B]{n \,\|\, \text{Alice} \,\|\, \text{Bob} \,\|\, \{ r_1 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob} \} k_A \,\|}$ Bob

Cathy $\xrightarrow{n \,\|\, \{ r_1 \,\|\, k_s \} k_A \,\|\, \{ r_2 \,\|\, k_s \} k_B}$ Bob

Alice $\xleftarrow{n \,\|\, \{ r_1 \,\|\, k_s \} k_A}$ Bob

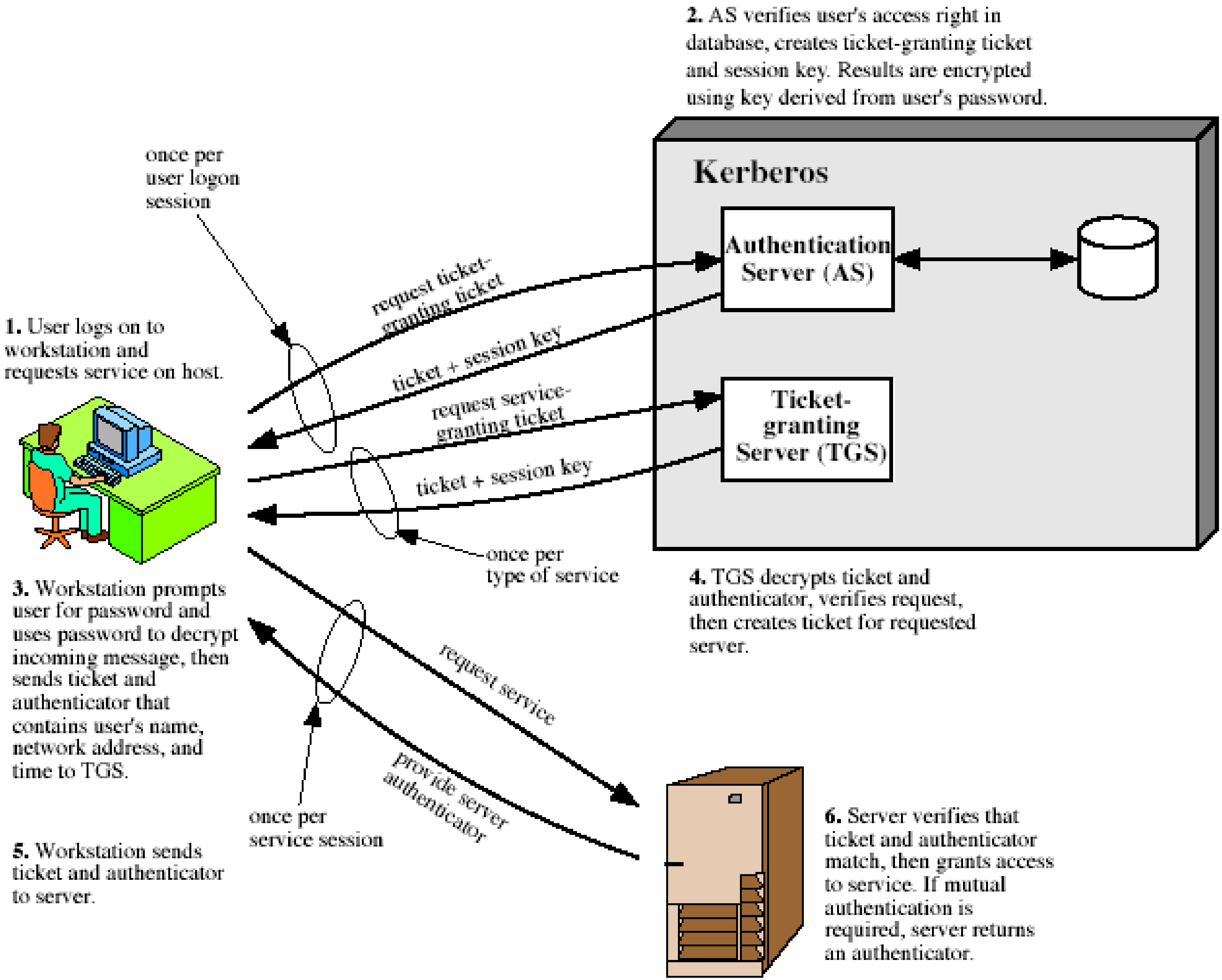
Uses integer *n* to associate all messages with a particular exchange

Replay Attack

- Eve acquires old k_s, message in third step
  - $n \,||\, \{ r_1 \,||\, k_s \} k_A \,||\, \{ r_2 \,||\, k_s \} k_B$
- Eve forwards appropriate part to Alice
  - If Alice has no ongoing key exchange with Bob
    - Accept/reject the message ?
  - Alice has ongoing key exchange with Bob
    - Accept/reject the message ?
- If replay is for the current key exchange, *and* Eve sent the relevant part *before* Bob did,
  - Does replay attack occur?

Kerberos

- Authentication system
  - Based on Needham-Schroeder with Denning-Sacco modification
  - Central server plays role of trusted third party ("Cathy")
- Ticket (credential)
  - Issuer vouches for identity of requester of service
- Authenticator
  - Identifies sender


2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
once per type of service
1. User logs on to workstation and requests service on host.
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
request service
provide server authenticator
once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Ticket

- Credential saying issuer has identified ticket requester
- Example ticket issued to user *u* for service *s*

  $T_{u,s} = s \,||\, \{\, u \,||\, u\text{'s address} \,||\, \text{valid time} \,||\, k_{u,s} \,\}k_s$

  where:
  - $k_{u,s}$ is session key for user and service
  - Valid time is interval for which the ticket is valid
  - *u*'s address may be IP address or something else
    - Note: more fields, but not relevant here

Authenticator

- Credential containing identity of sender of ticket
  - Used to confirm sender is entity to which ticket was issued
- Example: authenticator user *u* generates for service *s*

$$A_{u,s} = \{ u \,||\, \text{generation time} \,||\, k_t \}\, k_{u,s}$$

where:

  - k_t is alternate session key
  - Generation time is when authenticator generated
    - Note: more fields, not relevant here

Protocol

Authentication server

$$user \xrightarrow{\quad user \,\|\, TGS \quad} \mathrm{AS}$$

$$user \xleftarrow{\quad \{ k_{u,TGS} \} k_u \,\|\, T_{u,TGS} \quad} \mathrm{AS}$$

$$user \xrightarrow{\quad service \,\|\, A_{u,TGS} \,\|\, T_{u,TGS} \quad} \mathrm{TGS}$$

$$user \xleftarrow{\quad user \,\|\, \{ k_{u,s} \} k_{u,TGS} \,\|\, T_{u,s} \quad} \mathrm{TGS}$$

$$user \xrightarrow{\quad A_{u,s} \,\|\, T_{u,s} \quad} service$$

$$user \xleftarrow{\quad \{ t+1 \} k_{u,s} \quad} service$$

Problems

- Relies on synchronized clocks
  - If not synchronized and old tickets, authenticators not cached, replay is possible
- Tickets have some fixed fields
  - Dictionary attacks possible
  - Kerberos 4 session keys weak (had much less than 56 bits of randomness); researchers at Purdue found them from tickets in minutes

Public Key Key Exchange

- Here interchange keys known
  - e_A, e_B Alice and Bob's public keys known to all
  - d_A, d_B Alice and Bob's private keys known only to owner
- Simple protocol
  - k_s is desired session key

Problem and Solution?

$$\text{Alice} \xrightarrow{\{\, k_s \,\}\, e_B} \text{Bob}$$

Any problem ?

$$\text{Alice} \xrightarrow{\{\, \{\, k_s \,\}\, d_A \,\}\, e_B} \text{Bob}$$

What about this?

Public Key Key Exchange

- Assumes Bob has Alice's public key, and *vice versa*
  - If not, each must get it from public server
  - If keys not bound to identity of owner, attacker Eve can launch a *man-in-the-middle* attack

Man-in-the-Middle Attack

Alice $\xrightarrow{\text{send me Bob's public key}}$ | $\xrightarrow{\textit{Eve intercepts request}}$ Peter

Eve $\xrightarrow{\text{send me Bob's public key}}$ Peter

Eve $\xleftarrow{e_B}$ Peter

Alice $\xleftarrow{e_E}$ Eve

Alice $\xrightarrow{\{ k_s \}\ e_E}$ | $\xrightarrow{\textit{Eve intercepts message}}$ Bob

Eve $\xrightarrow{\{ k_s \}\ e_B}$ Bob

Peter is public server providing public keys

Cryptographic Key Infrastructure

- Goal:
  - bind identity to key
- Classical Crypto:
  - Not possible as all keys are shared
- Public key Crypto:
  - Bind identity to public key
  - Erroneous binding means no secrecy between principals
  - Assume principal identified by an acceptable name

Certificates

- Create token (message) containing
  - Identity of principal (here, Alice)
  - Corresponding public key
  - Timestamp (when issued)
  - Other information (identity of signer)

signed by trusted authority (here, Cathy)

$$C_A = \{ e_A \mid\mid \text{Alice} \mid\mid T \}\, d_C$$

C_A is A's certificate

Use

- Bob gets Alice’s certificate
  - If he knows Cathy’s public key, he can decipher the certificate
  - Now Bob has Alice’s public key
- Problem:
  - Bob needs Cathy’s public key to validate certificate
  - Two approaches:
    - Merkle’s tree, **Signature chains**

Certificate Signature Chains

- Create certificate
  - Generate hash of certificate
  - Encipher hash with issuer’s private key
- Validate
  - Obtain issuer’s public key
  - Decipher enciphered hash
  - Re-compute hash from certificate and compare
- Problem:
  - Validating the certificate of the issuer and getting issuer’s public key

X.509 Chains

- Key certificate fields in X.509v3:
  - Version
  - Serial number (unique)
  - Signature algorithm identifier
  - Issuer's name; uniquely identifies issuer
  - Interval of validity
  - Subject's name; uniquely identifies subject
  - Subject's public key info

  …
  - Signature:
    - Identifies algorithm used to sign the certificate
    - Signature (enciphered hash)

X.509 Certificate Validation

- Obtain issuer's public key
  - The one for the particular signature algorithm
- Decipher signature
  - Gives hash of certificate
- Re-compute hash from certificate and compare
  - If they differ, there's a problem
- Check interval of validity
  - This confirms that certificate is current

Issuers

- *Certification Authority (CA)*: entity that issues certificates
  - Multiple issuers pose validation problem
  - Alice’s CA is Cathy; Bob’s CA is Dan; how can Alice validate Bob’s certificate?
  - Have Cathy and Don cross-certify
    - Each issues certificate for the other

Validation and Cross-Certifying

- Certificates:
  - Cathy<<Alice>>
    - represents the certificate that C has generated for A
  - Dan<<Bob> ; Cathy<<Dan>>; Dan<<Cathy>>

- Alice validates Bob’s certificate
  - Alice obtains Cathy<<Dan>>
  - Can Alice validate Cathy<<Dan>> ? (how?)
  - Can Alice use Cathy<<Dan>> to validate Dan<<Bob>> ? (how?)
    - Signature chain : ??

  - Show how Bob can validate Alice’s certificate?

PGP Chains

- Pretty Good Privacy:
  - Widely used to provide privacy for electronic mail and signing files digitally
- OpenPGP certificates structured into packets
  - One public key packet
  - Zero or more signature packets

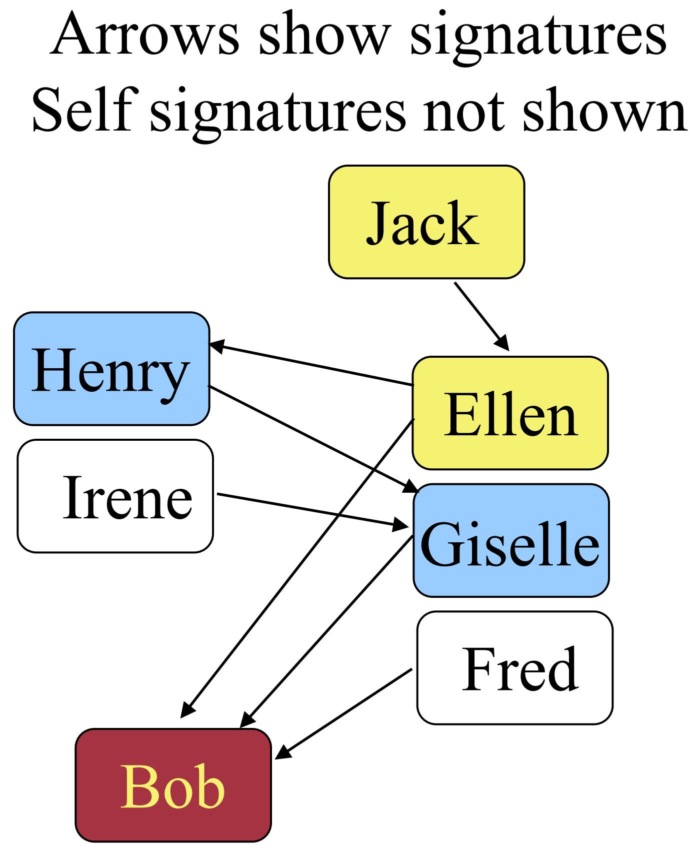
- Public key packet:
  - Version (3 or 4; 3 compatible with all versions of PGP, 4 not compatible with older versions of PGP)
  - Creation time
  - Validity period (not present in version 3)
  - Public key algorithm, associated parameters
  - Public key

OpenPGP Signature Packet

- Version 3 signature packet
  - Version (3)
  - Signature type (level of trust)
  - Creation time (when next fields hashed)
  - Signer’s key identifier (identifies key to encipher hash)
  - Public key algorithm (used to encipher hash)
  - Hash algorithm
  - Part of signed hash (used for quick check)
  - Signature (enciphered hash using signer’s private key)

Validating Certificates

- Alice needs to validate Bob's OpenPGP cert
  - Does not know Fred, Giselle, or Ellen
- Alice gets Giselle's cert
  - Knows Henry slightly, but his signature is at "casual" level of trust
- Alice gets Ellen's cert
  - Knows Jack, so uses his cert to validate Ellen's, then hers to validate Bob's



Digital Signature

- Construct that authenticates origin, contents of message in a manner provable to a disinterested third party (“judge”)
- Sender cannot deny having sent message
  - Limited to *technical* proofs
    - Inability to deny one’s cryptographic key was used to sign
  - One could claim the cryptographic key was stolen or compromised
    - Legal proofs, *etc.*, probably required;

Signature

- Classical: Alice, Bob share key *k*
  - Alice sends *m* || { *m* }*k* to Bob
  - Does this satisfy the requirement for message authentication? How?
  - Does this satisfy the requirement for a digital signature?

Classical Digital Signatures

- Require trusted third party
  - Alice, Bob share keys with trusted party Cathy
- The judge must trust Cathy

$$\text{Alice} \xrightarrow{\{m\}k_{Alice}} \text{Bob}$$

$$\text{Bob} \xrightarrow{\{m\}k_{Alice}} \text{Cathy}$$

$$\text{Cathy} \xrightarrow{\{m\}k_{Bob}} \text{Bob}$$

How can the judge resolve any dispute where one claims that the contract was not signed?

Public Key Digital Signatures (RSA)

- Alice's keys are d_{Alice}, e_{Alice}
- Alice sends Bob

$$m \,||\, \{ m \} d_{Alice}$$

- In case of dispute, judge computes

$$\{ \{ m \} d_{Alice} \} e_{Alice}$$

- and if it is *m*, Alice signed message
  - She's the only one who knows d_{Alice}!

RSA Digital Signatures

- Use private key to encipher message
  - Protocol for use is *critical*
- Key points:
  - Never sign random documents, and when signing, always sign hash and never document
    - Mathematical properties can be turned against signer
  - Sign message first, then encipher
    - Changing public keys causes forgery

Attack #1

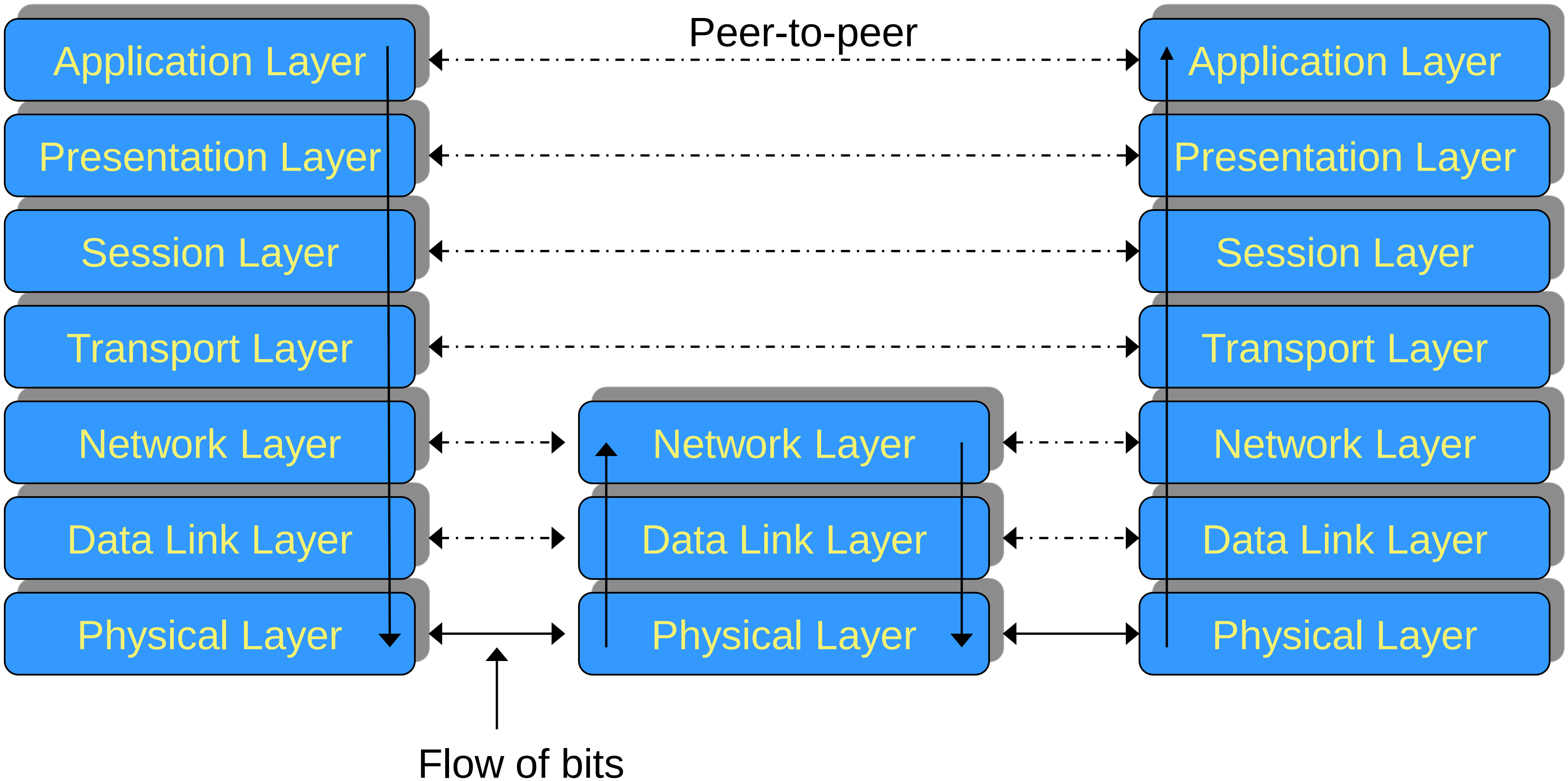
- Example: Alice, Bob communicating
  - $n_A = 95$, $e_A = 59$, $d_A = 11$
  - $n_B = 77$, $e_B = 53$, $d_B = 17$
- 26 contracts, numbered 00 to 25
  - Alice has Bob sign 05 and 17:
    - $c = m^{d_B} \bmod n_B = 05^{17} \bmod 77 = 3$
    - $c = m^{d_B} \bmod n_B = 17^{17} \bmod 77 = 19$
  - Alice computes $05 \times 17 \bmod 77 = 08$; corresponding signature is $03 \times 19 \bmod 77 = 57$; claims Bob signed 08

    Note: $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$
  - Judge computes $c^{e_B} \bmod n_B = 57^{53} \bmod 77 = 08$
    - Signature validated; Bob is toast!

Attack #2: Bob's Revenge

- Bob, Alice agree to sign contract 06
- Alice enciphers, then signs:
  - Enciper: $c = m^{e_B} \bmod n_B = 06^{53} \bmod 77$
  - Sign: $c^{d_A} \bmod n_A = (06^{53} \bmod 77)^{11} \bmod 95 = 63$
- Bob now changes his public key
  - Bob wants to claim that Alice singed N (13)
  - Computes r such that $13^r \bmod 77 = 6$; say, $r = 59$
  - Computes $r.e_B \bmod \varphi(n_B) = 59 \times 53 \bmod 60 = 7$
  - Replace public key e_B with 7, private key $d_B = 43$
- Bob claims contract was 13. Judge computes:
  - $(63^{59} \bmod 95)^{43} \bmod 77 = 13$
  - Verified; now Alice is toast
- Solution: sign first and then encipher!!

ISO/OSI Model



Security at the Transport Layer Secure Socket Layer (SSL)

- Developed by Netscape to provide security in WWW browsers and servers
- SSL is the basis for the Internet standard protocol – Transport Layer Security (TLS) protocol (compatible with SSLv3)
- Key idea: *Connections* and *Sessions*
  - A SSL session is an association between two peers
  - An SSL connection is the set of mechanisms used to transport data in an SSL session

Secure Socket Layer (SSL)

- Each party keeps session information
  - Session identifier (unique)
  - The peer's X.503(v3) certificate
  - Compression method used to reduce volume of data
  - Cipher specification (parameters for cipher and MAC)
  - Master secret of 48 bits
- Connection information
  - Random data for the server & client
  - Server and client keys (used for encryption)
  - Server and client MAC key
  - Initialization vector for the cipher, if needed
  - Server and client sequence numbers
- Provides a set of supported cryptographic mechanisms that are setup during negotiation (handshake protocol)

SSL Architecture

Provides a basis for
Secure communication
Confidentiality +
Message authenticity

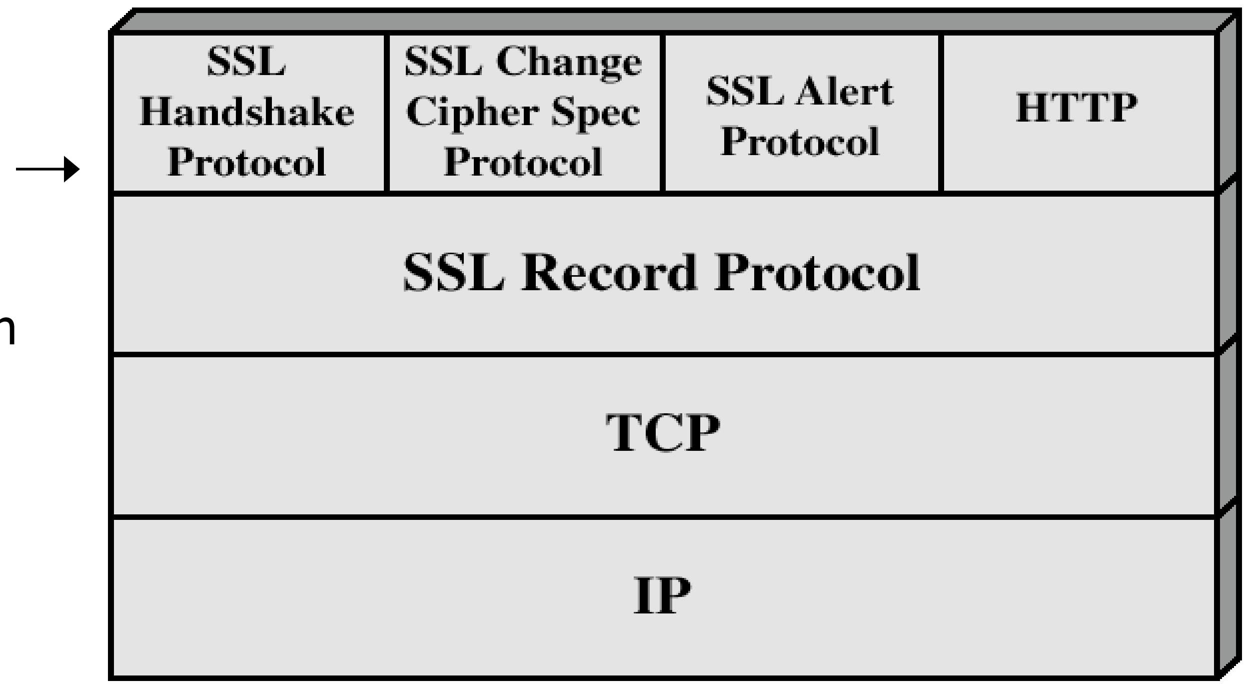
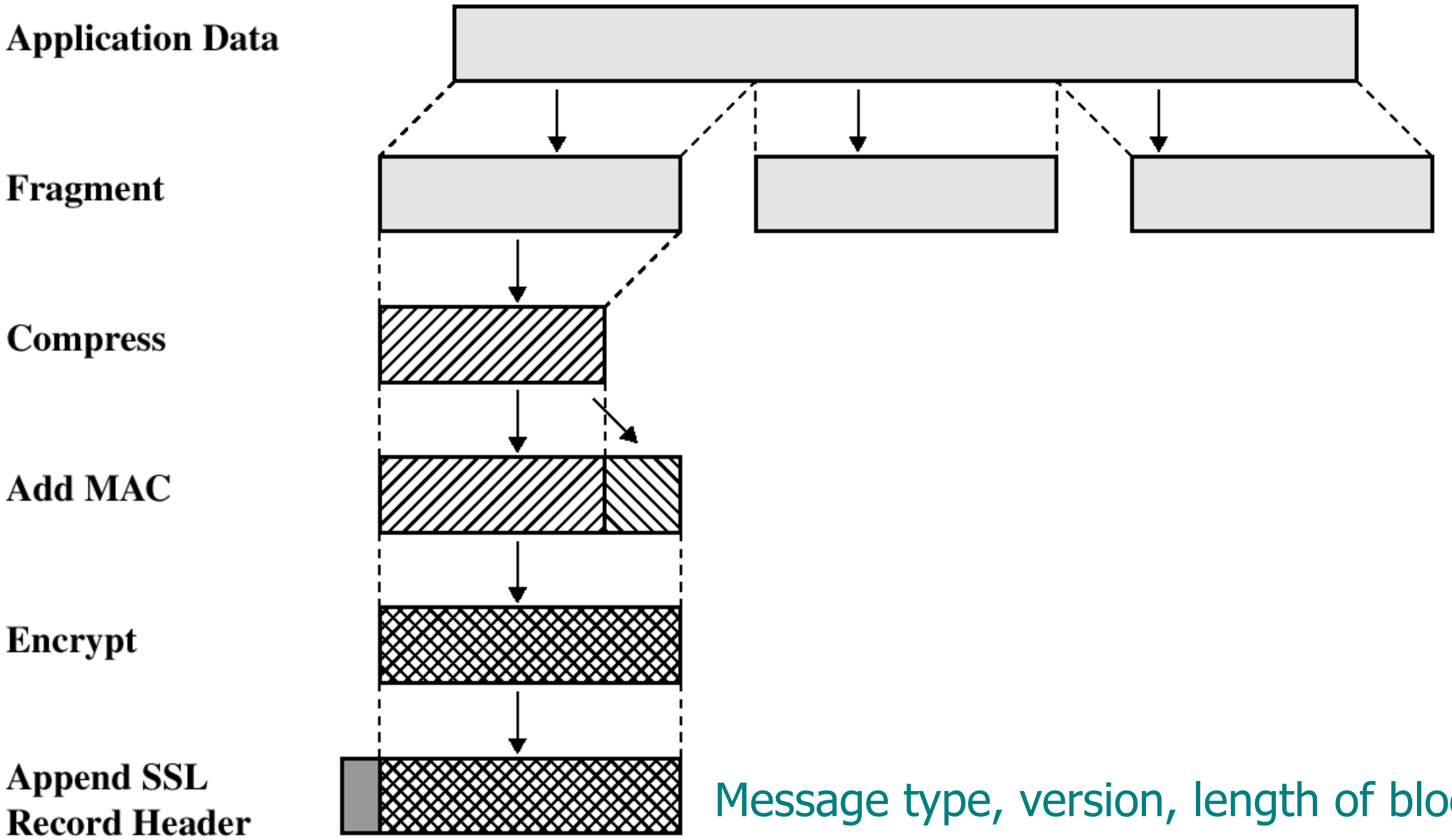


Figure 7.2 SSL Protocol Stack

SSL Record Protocol Operation

e.g., HTTP messages



Message type, version, length of block

Handshake Protocol

- The most complex part of SSL
- Allows the server and client to authenticate each other
  - Based on interchange cryptosystem (e.g., RSA)
- Negotiate encryption, MAC algorithm and cryptographic keys
  - Four rounds
- Used before any application data are transmitted

Other protocols

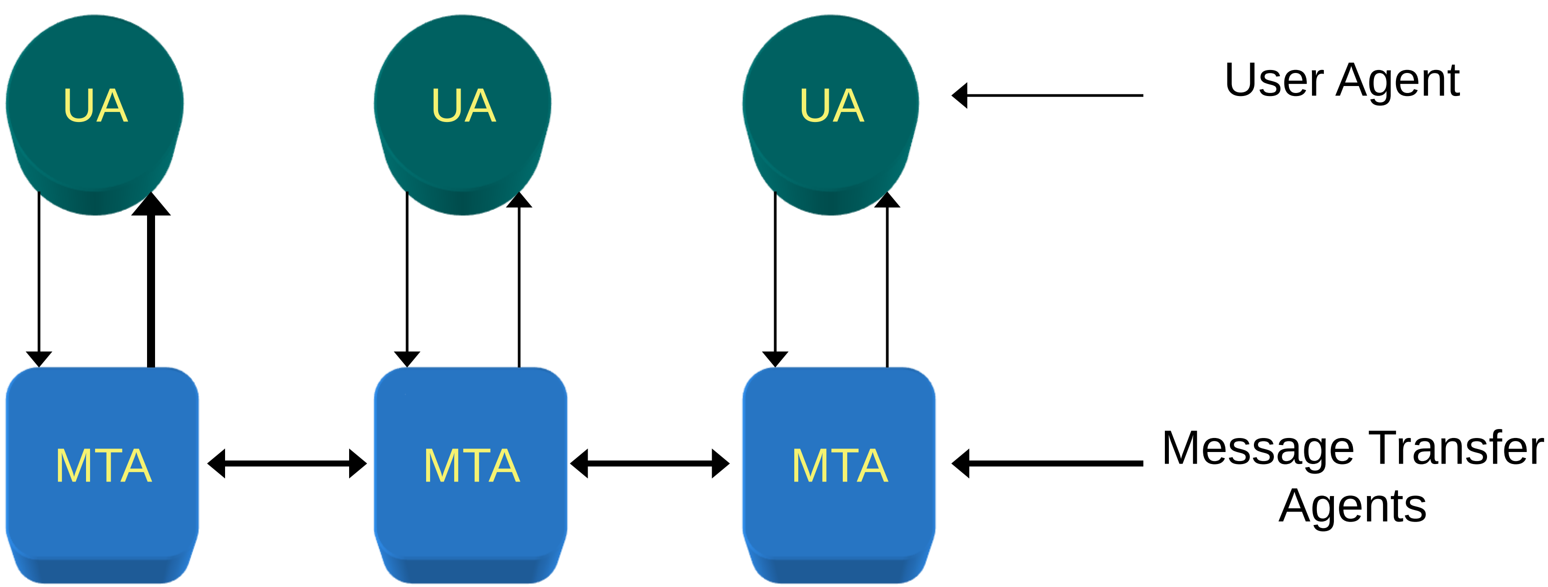
- SSL Change Cipher Spec Protocol
  - A single byte is exchanged
  - After new cipher parameters have been negotiated (renegotiated)
- SSL Alert Protocol
  - Signals an unusual condition
  - *Closure alert* : sender will not send anymore
  - *Error alert*: fatal error results in disconnect

Protocols

- End-to-end protocol
  - Example: telnet
- End-to-end encryption
  - Example: telnet with messages encrypted/decrypted at the client and server
  - Attackers on the intermediate hosts cannot read the message
- Link protocol
  - Protocol between every directly connected systems
    - Example: IP – guides messages from a host to one of its immediate host
- Link encryption
  - Encipher messages between intermediate host
  - Each host share a cryptographic key with its neighbor
    - Attackers at the intermediate host will be able to read the message

Electronic Mail

- UA interacts with the sender
- UA hands it to a MTA
- Attacker can read email on any of the computer with MTA
- Forgery possible



Security at the Application Layer: Privacy-enhanced Electronic Mail

- Study by Internet Research Task Force on Privacy or Privacy Research Group to develop protocols with following services
  - Confidentiality, by making the message unreadable except to the sender and recipients
  - Origin authentication, by identifying the sender precisely
  - Data integrity, by ensuring that any changes In the message are easy to detect
  - Non-repudiation of the origin (if possible)

Design Considerations/goals for PEM

- Not to redesign existing mail system protocols
- To be compatible with a range of MTAs, UAs and other computers
- To make privacy enhancements available separately so they are not required
- To enable parties to use the protocol to communicate without prearrangement

PEM
Basic Design

- Defines two keys
  - Data Encipherment Key (DEK) to encipher the message sent
    - Generated randomly
    - Used only once
    - Sent to the recipient
  - Interchange key: to encipher DEK
    - Must be obtained some other way than through the message

Protocols

- Confidential message (DEK: k_s)

$$\text{Alice} \xrightarrow{\{m\}k_s \,||\, \{k_s\}k_{\text{Bob}}} \text{Bob}$$

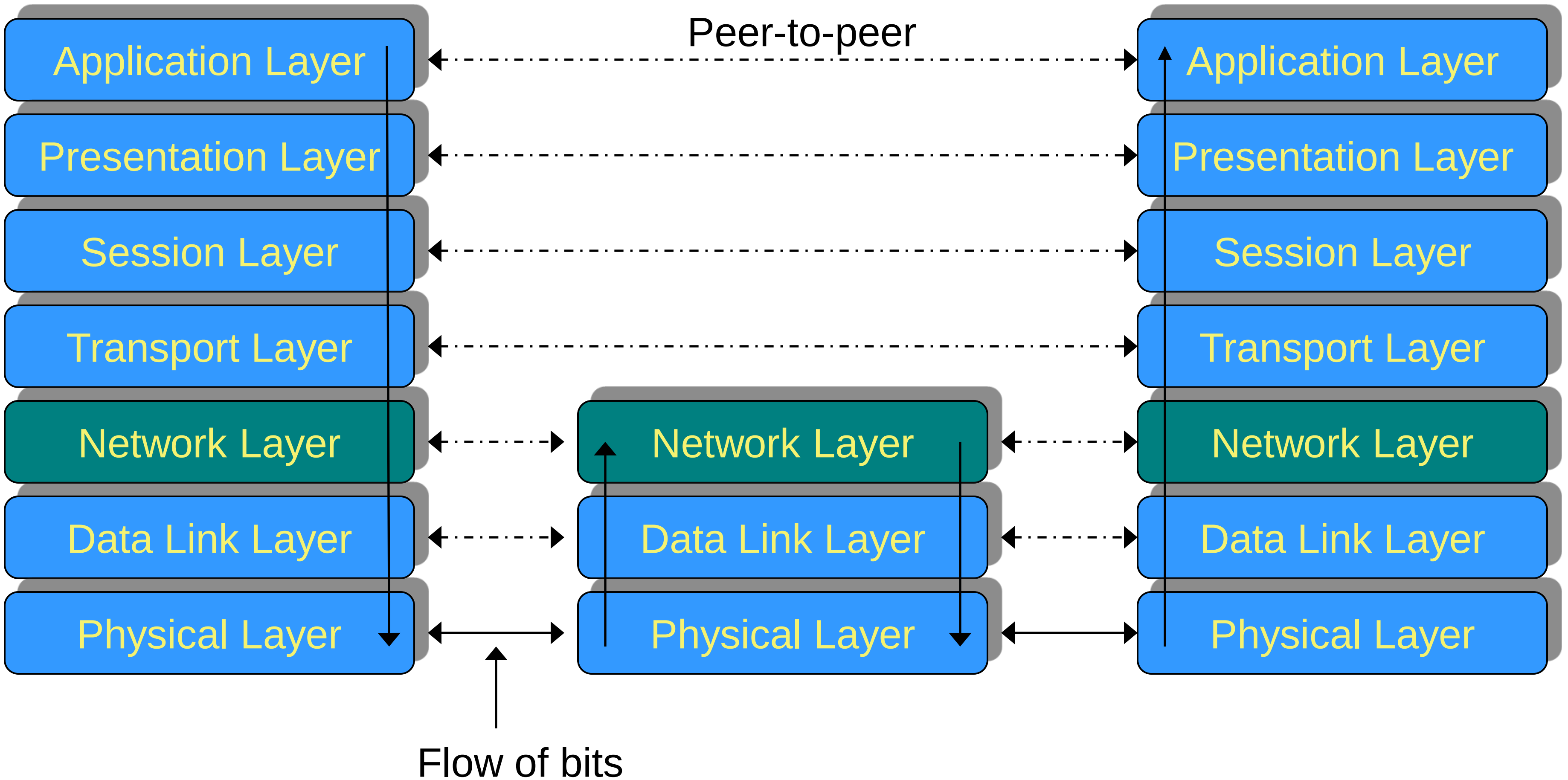
- Authenticated, integrity-checked message

$$\text{Alice} \xrightarrow{m \,||\, \{h(m)\}k_{\text{Alice}}} \text{Bob}$$

- Enciphered, authenticated, integrity checked message

$$\text{Alice} \xrightarrow{??} \text{Bob}$$

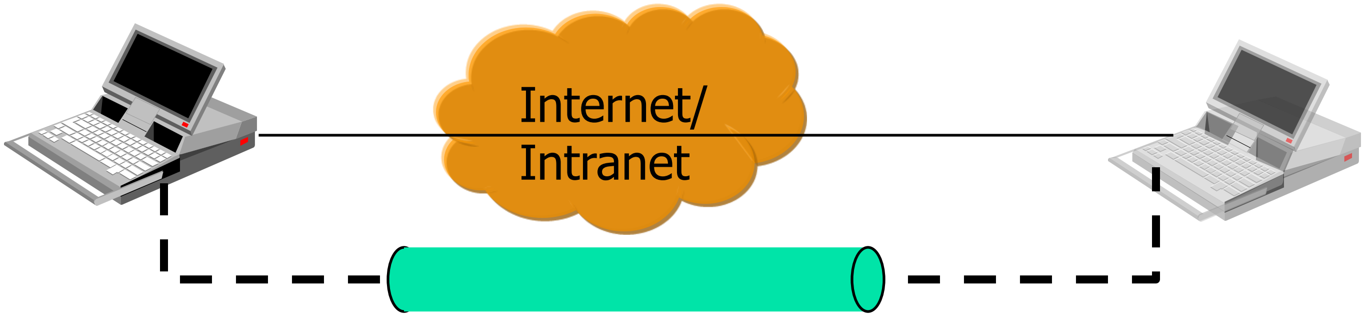
ISO/OSI Model
IPSec: Security at Network Layer



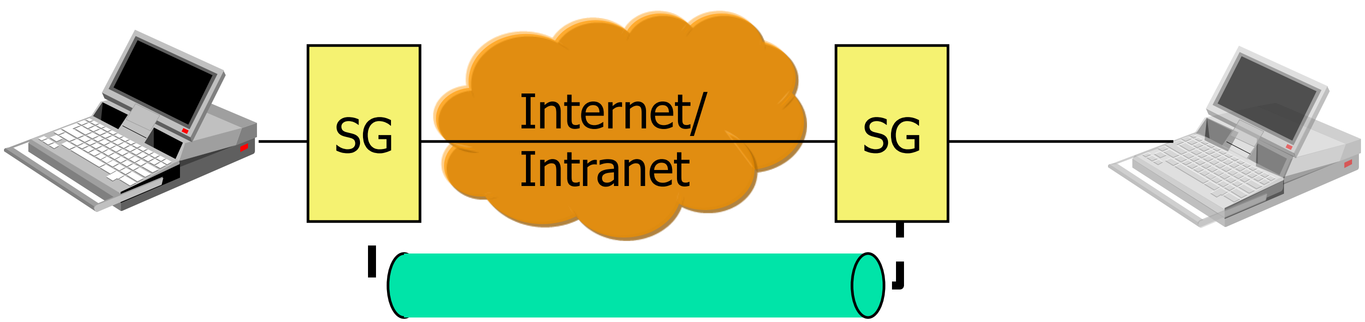
IPSec Protocols

- Authentication header (AH) protocol
  - Message integrity
  - Origin authentication
  - Anti-replay services
- Encapsulating security payload (ESP) protocol
  - Confidentiality
  - Message integrity
  - Origin authentication
  - Anti-replay services
- Internet Key Exchange (IKE)
  - Exchanging keys between entities that need to communicate over the Internet
  - What authentication methods to use, how long to use the keys, etc.

Cases where IPSec can be used

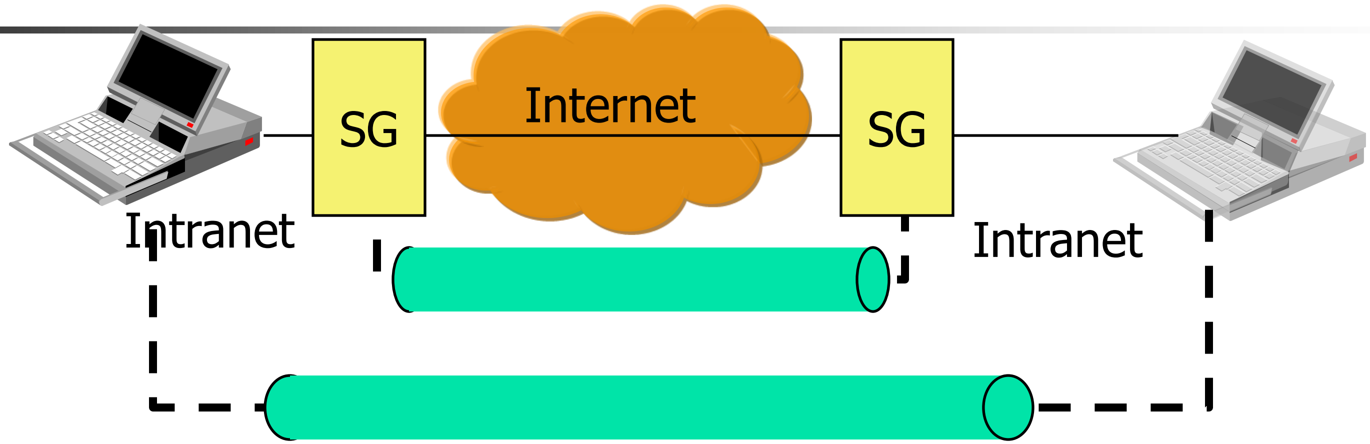


End-to-end security between two hosts

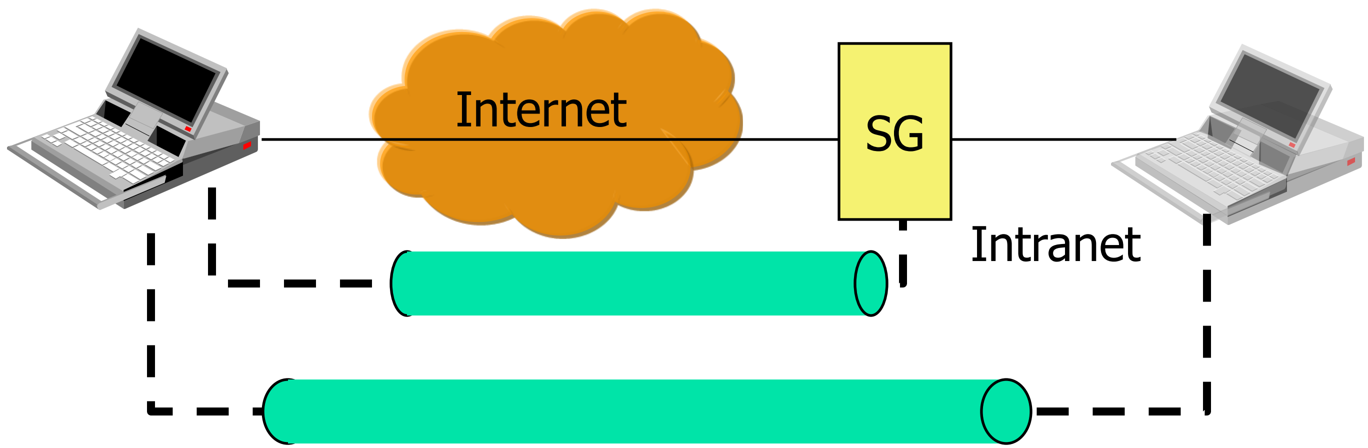


End-to-end security between two security gateways

Cases where IPSec can be used (2)



End-to-end security between two hosts + two gateways



End-to-end security between two hosts during dial-up

Security Association (SA)

- Unidirectional relationship between peers
- Specifies the security services provided to the traffic carried on the SA
  - Security enhancements to a channel along a path
- Identified by three parameters:
  - IP Destination Address
  - Security Protocol Identifier
    - Specifies whether AH or ESP is being used
  - Security Parameters Index (SPI)
    - Specifies the security parameters associated with the SA

Security Association (2)

- Each SA uses AH or ESP (not both)
  - If both required two SAs are created
- Multiple security associations may be used to provide required security services
  - A sequence of security associations is called *SA bundle*
  - Example: We can have an AH protocol followed by ESP or vice versa

Security Association Databases

- IP needs to know the SAs that exist in order to provide security services
- Security Policy Database (SPD)
  - IPSec uses SPD to handle messages
    - For each IP packet, it decides whether an IPSec service is provided, bypassed, or if the packet is to be discarded
- Security Association Database (SAD)
    - Keeps track of the sequence number
    - AH information (keys, algorithms, lifetimes)
    - ESP information (keys, algorithms, lifetimes, etc.)
    - Lifetime of the SA
    - Protocol mode
    - MTU et.c.

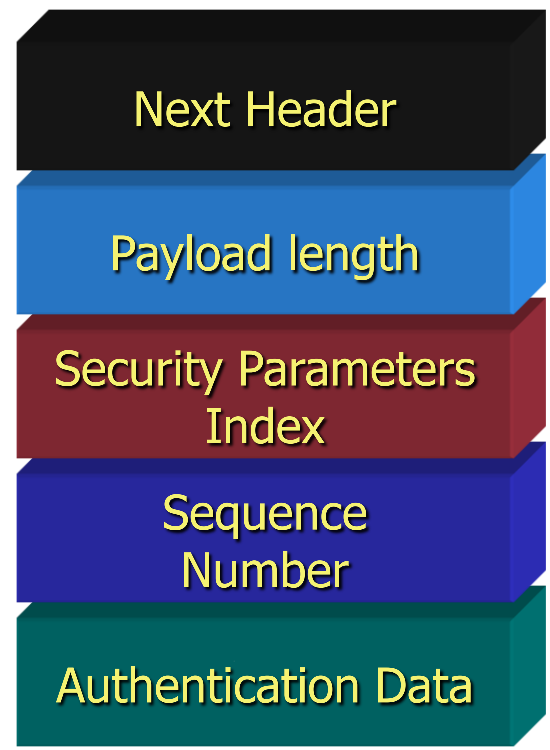
IPSec Modes

- Two modes
  - Transport mode
    - Encapsulates IP packet data area
    - IP Header is not protected
      - Protection is provided for the upper layers
      - Usually used in host-to-host communications
  - Tunnel mode
    - Encapsulates entire IP packet in an IPSec envelope
      - Helps against traffic analysis
      - The original IP packet is untouched in the Internet

Authentication Header (AH)

- Next header
  - Identifies what protocol header follows
- Payload length
  - Indicates the number of 32-bit words in the authentication header
- Security Parameters Index
  - Specifies to the receiver the algorithms, type of keys, and lifetime of the keys used
- Sequence number
  - Counter that increases with each IP packet sent from the same host to the same destination and SA
- Authentication Data

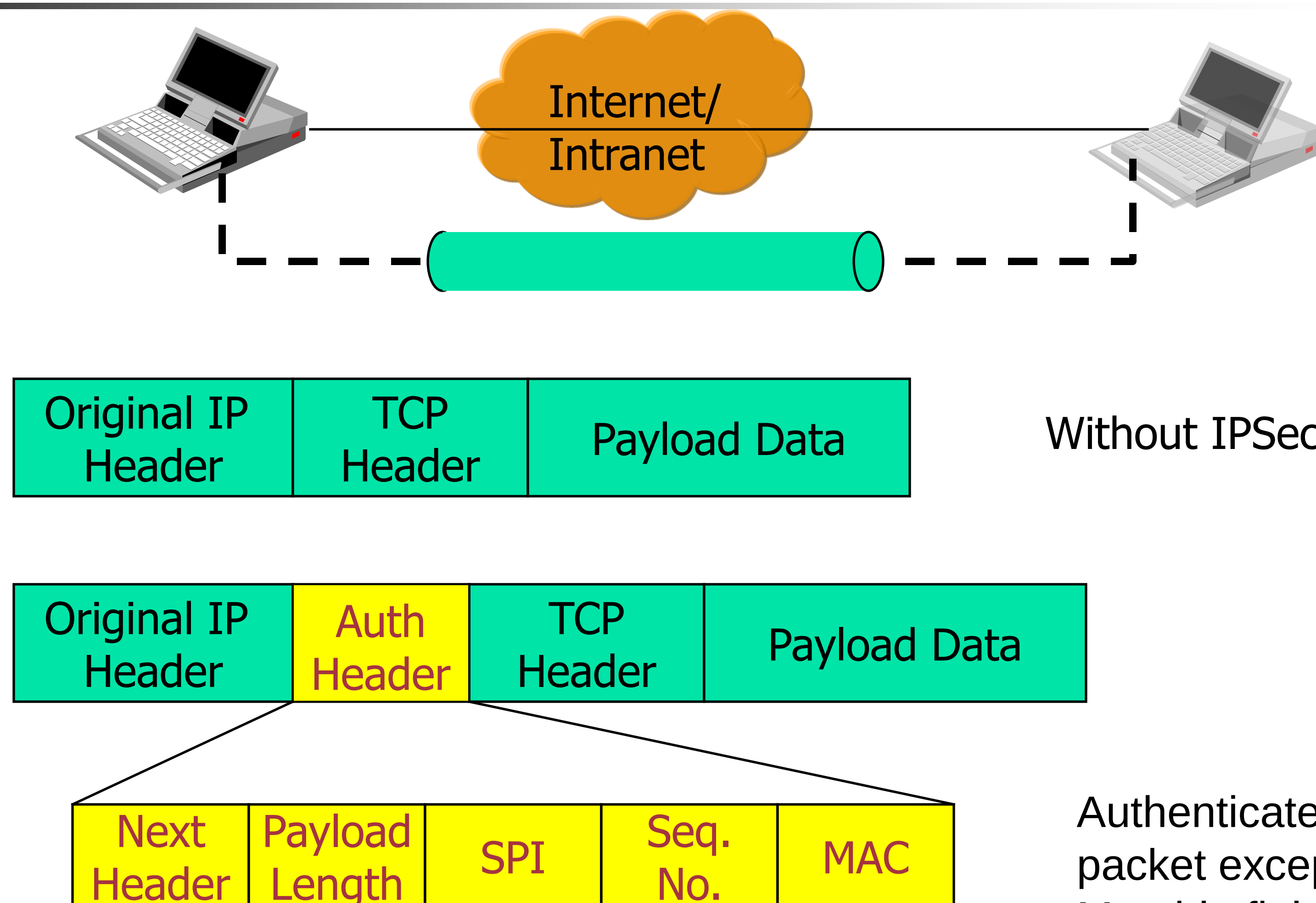
parameters



Preventing replay

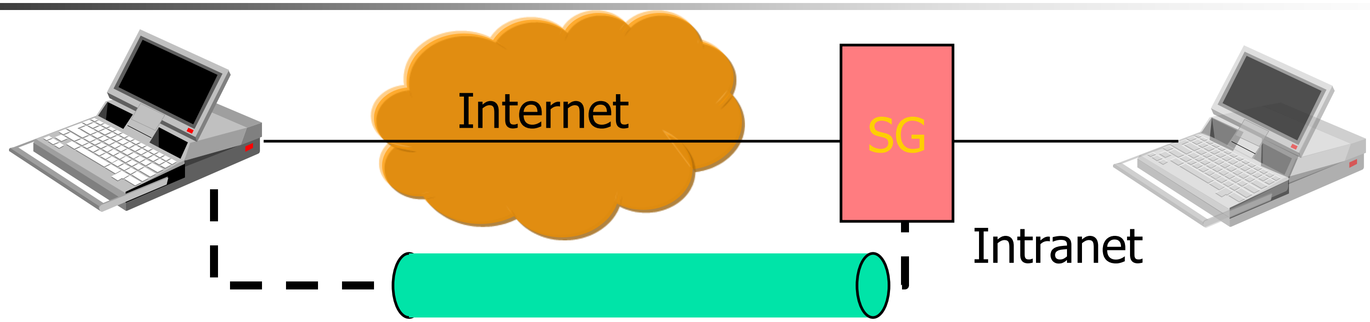
- Using 32 bit sequence numbers helps detect replay of IP packets
- The sender initializes a sequence number for every SA
- Receiver implements a window size of W to keep track of authenticated packets
- Receiver checks the MAC to see if the packet is authentic

Transport Mode AH



Authenticate Entire packet except for Mutable fields

Tunnel Mode AH



Original IP Header	TCP Header	Payload Data

Without IPSec

New IP Header	Auth Header	Original IP Header	TCP Header	Payload Data

Next Header	Payload Length	SPI	Seq. No.	MAC

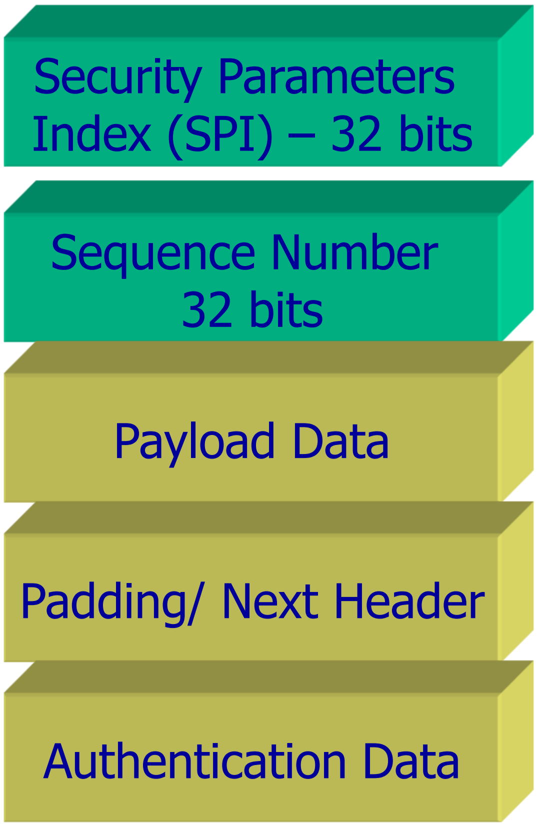
Authenticate
Entire IP Packet

ESP – Encapsulating Security Payload

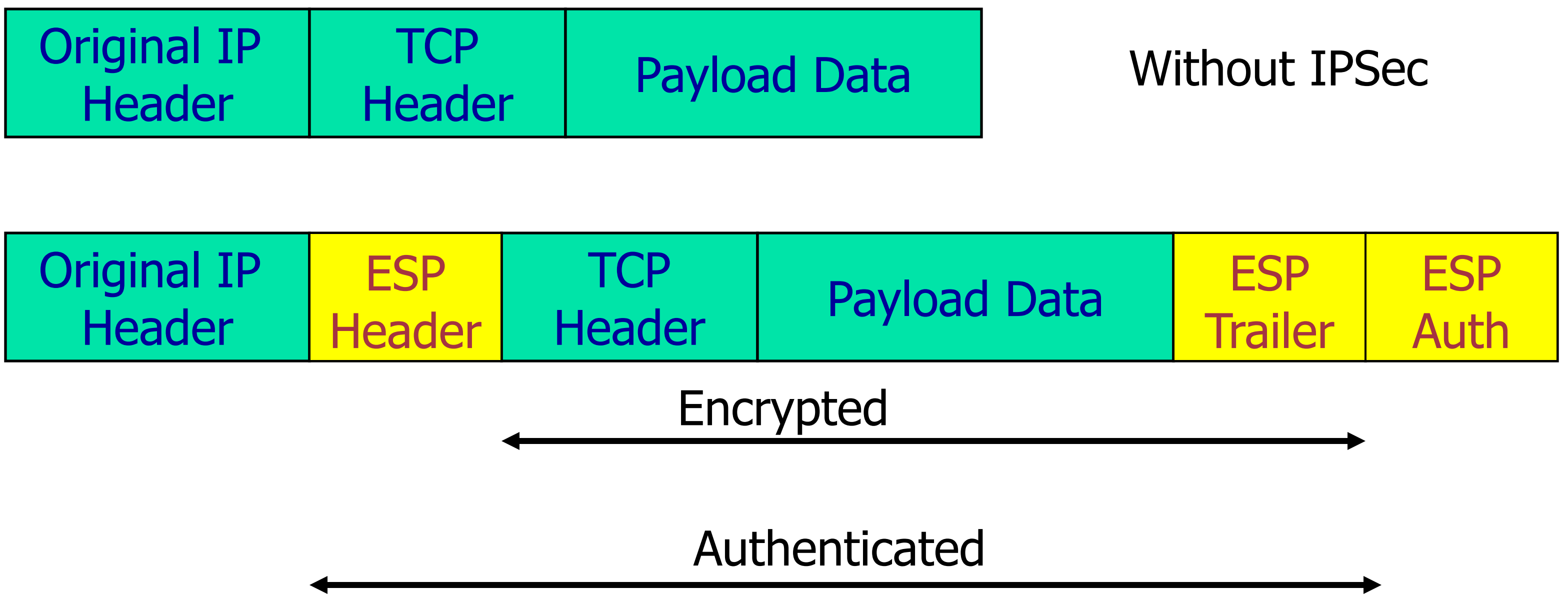
- Creates a new header in addition to the IP header
- Creates a new trailer
- Encrypts the payload data
- Authenticates
- Prevents replay

ESP – Encapsulating Security Payload

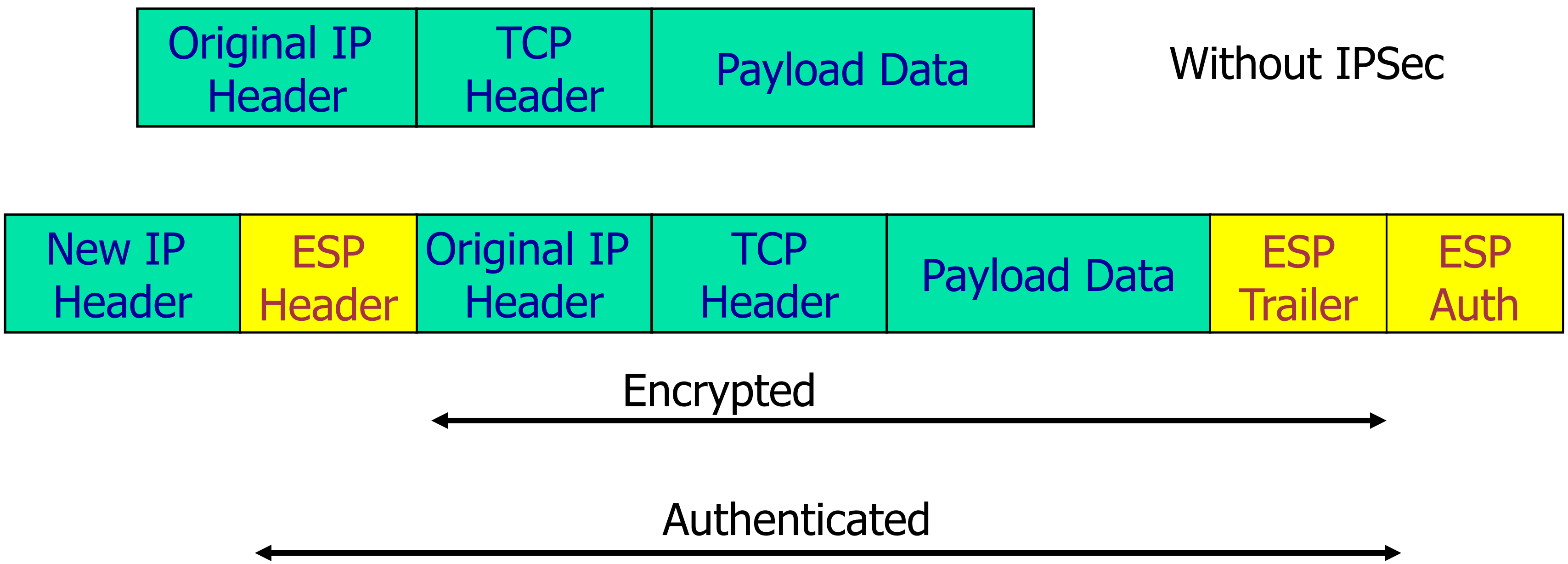
- Security Parameters Index (SPI)
  - Specifies to the receiver the algorithms, type of keys, and lifetime of the keys used
- Sequence number
  - Counter that increases with each IP packet sent from the same host to the same destination and SA
- Payload (variable)
  - TCP segment (transport mode) or IP packet (tunnel mode) - encryption
- Padding (+ Pad length, next Header)
  - 0 to 255 bytes of data to enable encryption algorithms to operate properly
- Authentication Data
  - MAC created over the packet



Transport mode ESP



Tunnel mode ESP



Summary

- Session key is better for secret message exchange
- Public key good for interchange key, digital signatures – needs certification system
- Various replay/MITM attacks are possible in key exchange protocols and care is needed
- Security services available at different levels