

# Introduction to Networking

CAN201 – Lecture 12

Lecturer: Dr. Wenjun Fan

# Lecture 12 – Network Security (3)

- **Roadmap**

1. Network layer security: IPsec
2. Operational security: firewalls and IDS



# What is network-layer confidentiality ?

***between two network entities:***

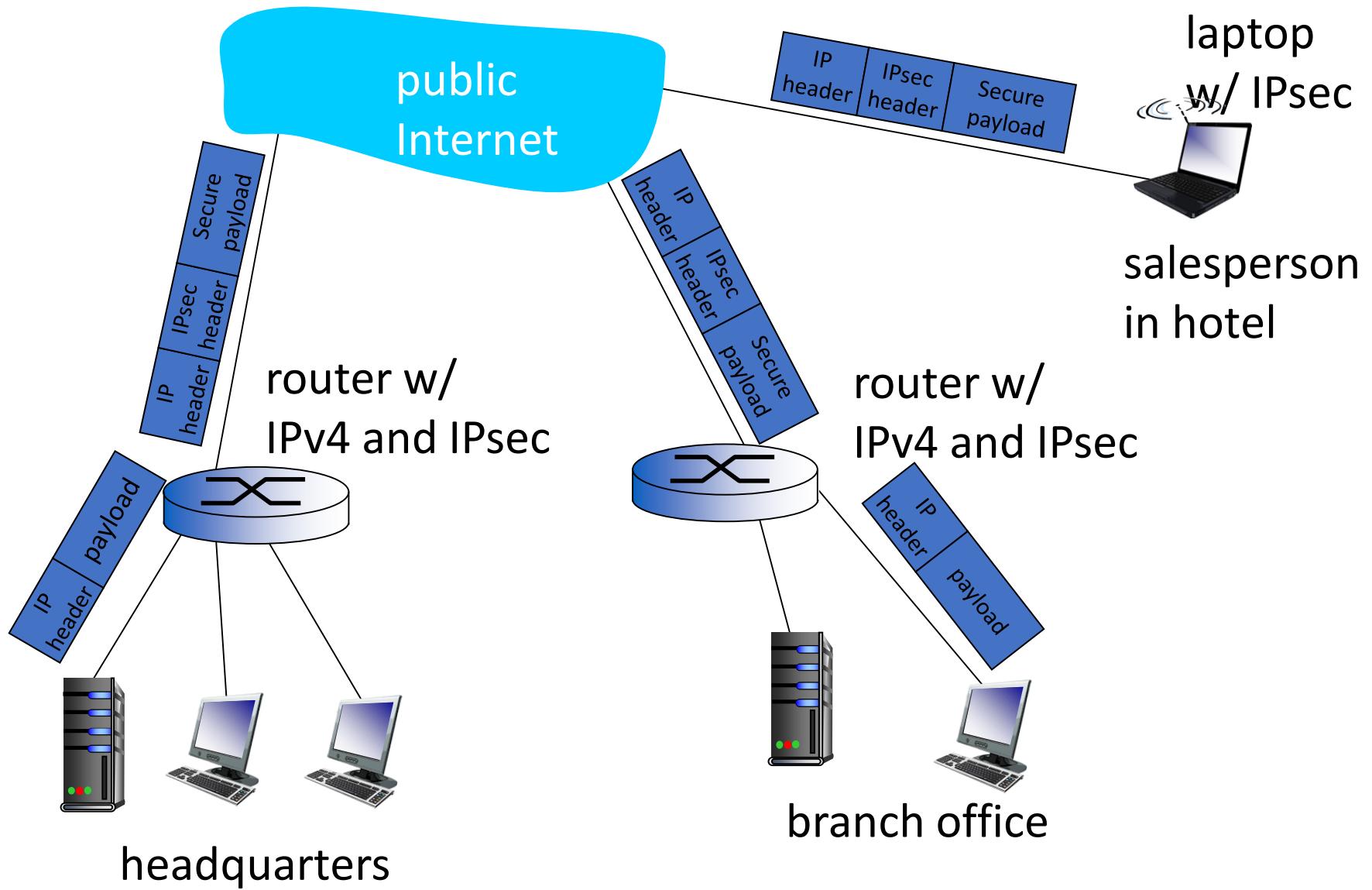
- sending entity encrypts datagram payload, payload could be:
  - TCP or UDP segment, ICMP message, OSPF message ...
- all data sent from one entity to other would be hidden from any third party (that presumably is sniffing the network):
  - web pages, e-mail, P2P file transfers, TCP SYN packets ...
- “blanket coverage”

# Virtual Private Networks (VPNs)

## ***motivation:***

- **institutions often want private networks for security.**
  - institution could actually deploy a stand-alone physical network that is completely separate from the public Internet.
  - costly: separate routers, links, DNS infrastructure.
- **VPN: institution's inter-office traffic is sent over public Internet instead**
  - encrypted before entering public Internet
  - logically separate from other traffic

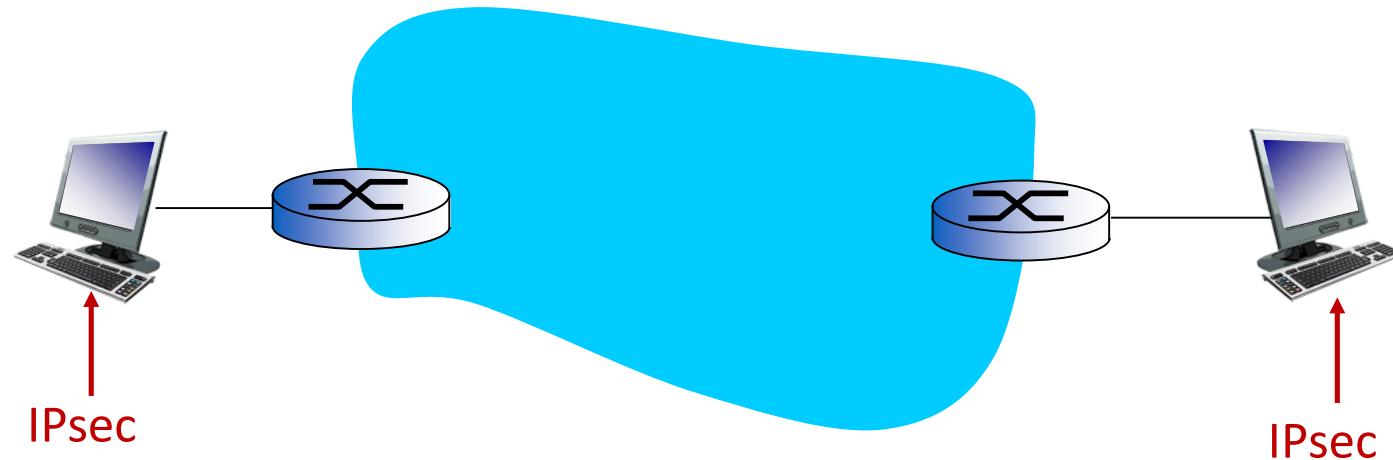
# Virtual Private Networks (VPNs)



# IPsec services

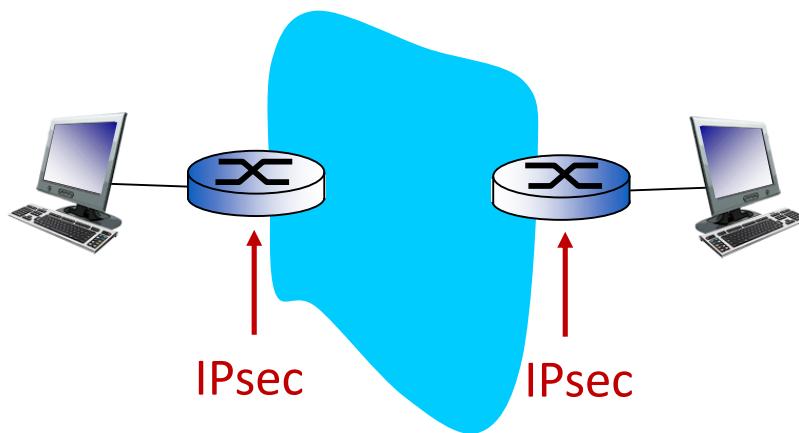
- **data integrity**
- **origin authentication**
- **replay attack prevention**
- **confidentiality**
- **two protocols providing different service models:**
  - **Authentication Header (AH)**
    - provides source authentication and data integrity but *no* confidentiality
  - **Encapsulation Security Payload (ESP)**
    - provides source authentication, data integrity, and confidentiality
    - more widely used than AH

# IPsec transport mode (host mode)

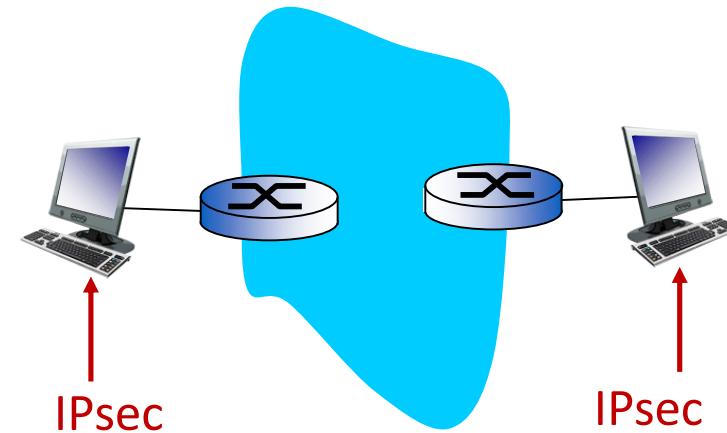


- IPsec datagram emitted and received by end-system
- protects upper level protocols

# IPsec – tunneling mode



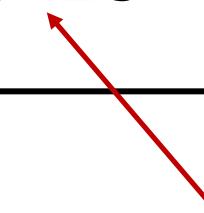
- edge routers IPsec-aware



- hosts IPsec-aware

# Four combinations are possible!

|                        |                         |
|------------------------|-------------------------|
| Host mode<br>with AH   | Host mode<br>with ESP   |
| Tunnel mode<br>with AH | Tunnel mode<br>with ESP |

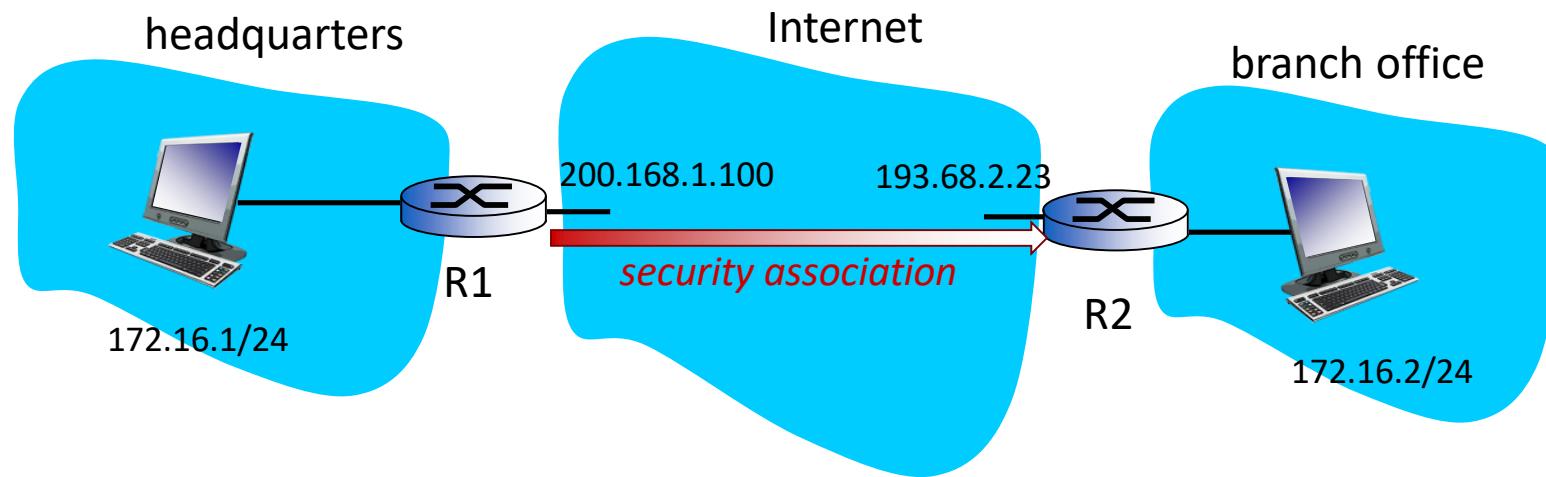


most common and  
most important

# Security associations (SAs)

- before sending data, “**security association (SA)**” established from sending to receiving entity
  - SAs are simplex: logical connection for only one direction
- **ending, receiving entities maintain state information about SA**
  - recall: TCP endpoints also maintain state info
  - IP is connectionless; IPsec is connection-oriented!
- **how many SAs in VPN w/ one headquarters office, one branch office, and n traveling salesperson?**

# Example SA from R1 to R2



## *R1 stores for SA:*

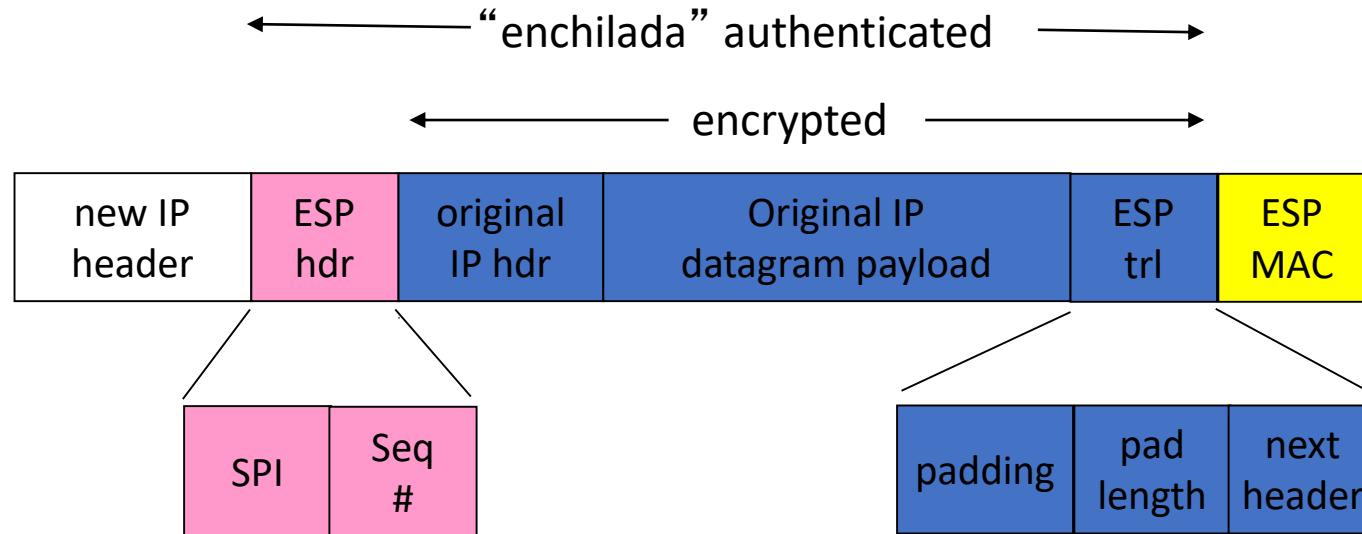
- 32-bit SA identifier: **Security Parameter Index (SPI)**
- origin SA interface (**200.168.1.100**)
- destination SA interface (**193.68.2.23**)
- type of encryption used (e.g., 3DES with CBC)
- encryption key
- type of integrity check used (e.g., HMAC with MD5)
- authentication key

# Security Association Database (SAD)

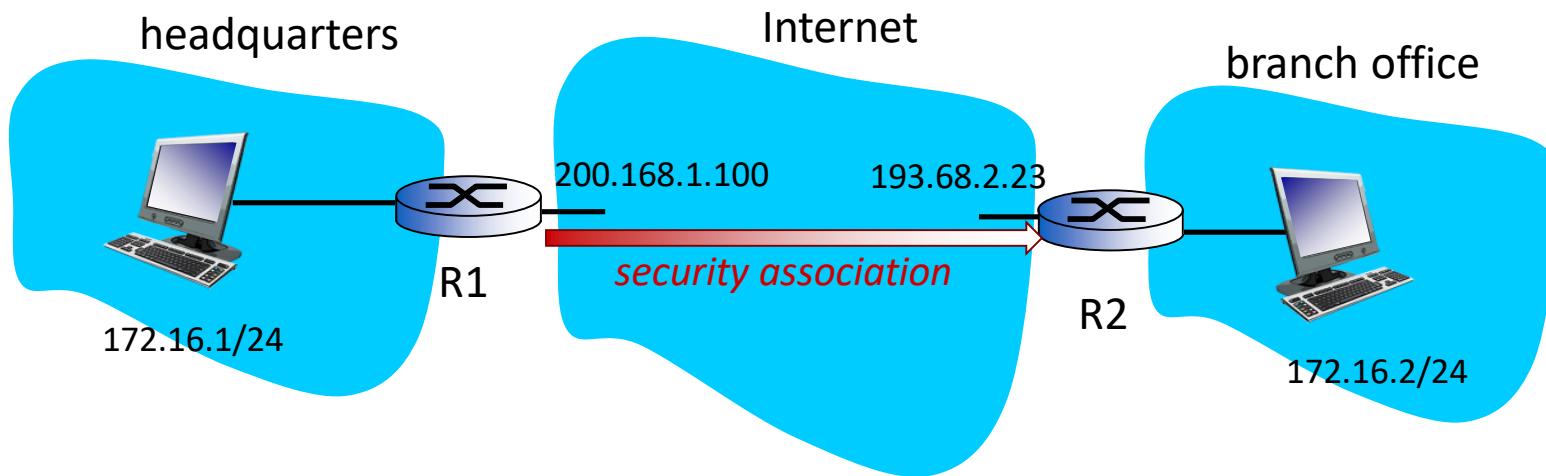
- endpoint holds SA state in *security association database (SAD)*, where it can locate them during processing.
- with n salespersons,  $2 + 2n$  SAs in R1's SAD
- when sending IPsec datagram, R1 accesses SAD to determine how to process datagram.
- when IPsec datagram arrives to R2, R2 examines **SPI** in IPsec datagram, indexes SAD with SPI, and processes datagram accordingly.

# IPsec datagram

focus for now on tunnel mode with ESP

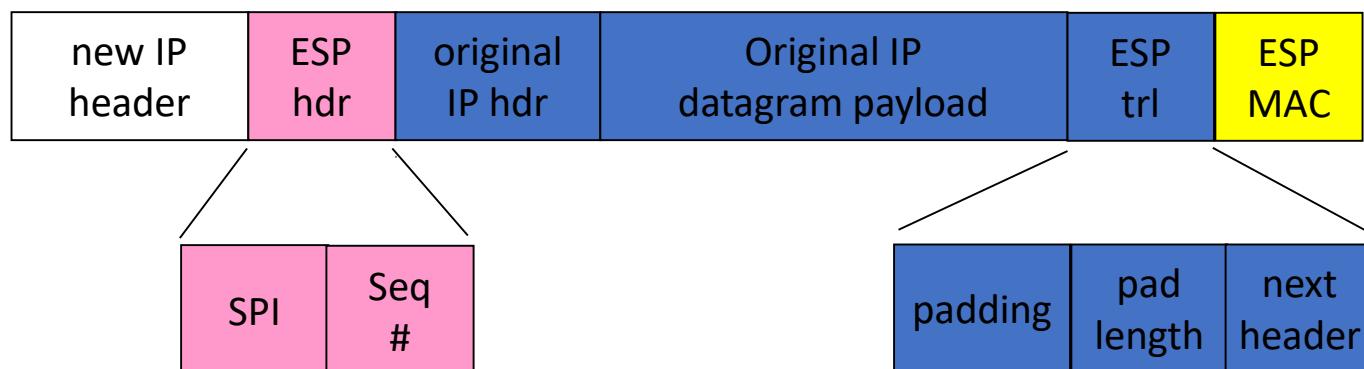


# What happens?



← “enchilada” authenticated →

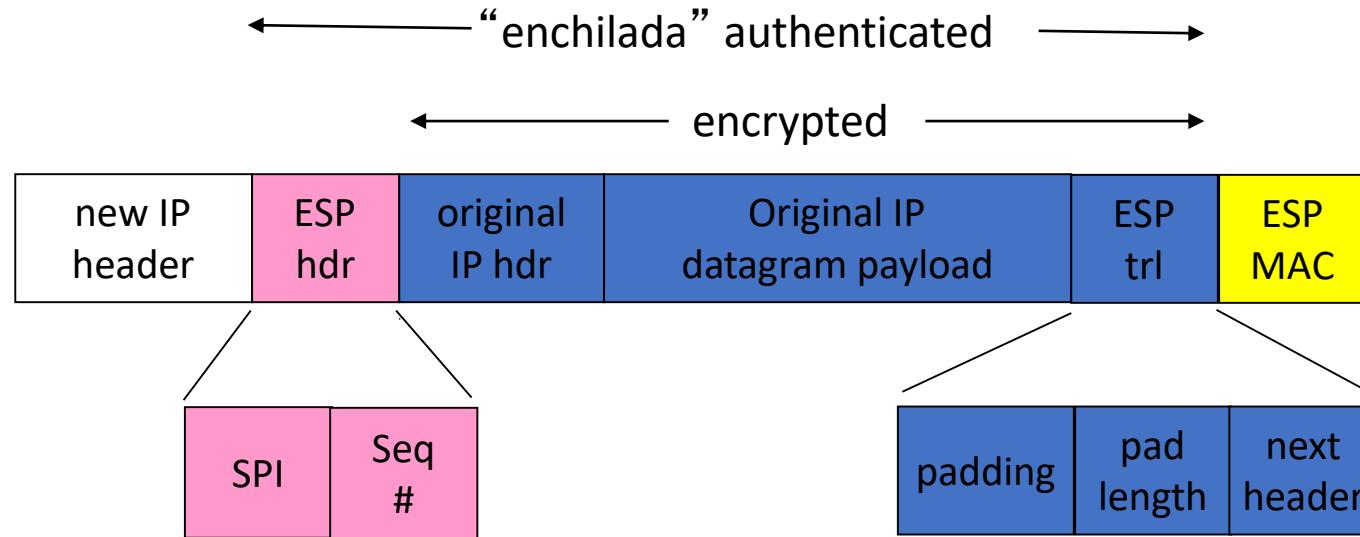
← encrypted →



# R1: convert original datagram to IPsec datagram

- appends to back of original datagram (that includes original header fields!) an “ESP trailer” field.
- encrypts result using algorithm & key specified by SA.
- appends to front of this encrypted quantity the “ESP header, creating “enchilada”.
- creates authentication MAC over the *whole enchilada*, using algorithm and key specified in SA.
- appends MAC to back of *enchilada*, forming *payload*.
- creates brand new IP header, with all the classic IPv4 header fields, which it appends before payload.

# Inside the enchilada:



- **ESP trailer: Padding for block ciphers**
- **ESP header:**
  - SPI, so receiving entity knows what to do
  - Sequence number, to thwart replay attacks
- **MAC in ESP auth field is created with shared secret key**

# IPsec sequence numbers

- **for new SA, sender initializes seq. # to 0**
- **each time datagram is sent on SA:**
  - sender increments seq # counter
  - places value in seq # field
- **goal:**
  - prevent attacker from sniffing and replaying a packet
  - receipt of duplicate, authenticated IP packets may disrupt service
- **method:**
  - destination checks for duplicates
  - doesn't keep track of *all* received packets; instead uses a window

# Security Policy Database (SPD)

- **policy:** For a given datagram, sending entity needs to know if it should use IPsec or vanilla IP
- needs also to know which SA to use
  - may use: source and destination IP address; protocol number
- info in SPD indicates “what” to do with arriving datagram
- info in SAD indicates “how” to do it

# Summary: IPsec services



- suppose Trudy sits somewhere between R1 and R2. she doesn't know the keys.
  - will Trudy be able to see original contents of datagram? How about source, dest IP address, transport protocol, application port?
  - flip bits without detection?
  - masquerade as R1 using R1's IP address?
  - replay a datagram?

# IKE: Internet Key Exchange

- ***previous examples:*** manual establishment of IPsec SAs in IPsec endpoints:

## *Example SA*

SPI: 12345

Source IP: 200.168.1.100

Dest IP: 193.68.2.23

Protocol: ESP

Encryption algorithm: 3DES-cbc

HMAC algorithm: MD5

Encryption key: 0x7aeaca...

HMAC key: 0xc0291f...

- manual keying is impractical for VPN with 100s of endpoints
- instead use ***IPsec IKE (Internet Key Exchange)*** protocol, specified in RFC 5996.

# IPsec summary

- IKE message exchange for algorithms, secret keys, SPI numbers
- either AH or ESP protocol (or both)
  - AH provides integrity, source authentication
  - ESP protocol (with AH) additionally provides encryption
- IPsec peers can be two end systems, two routers/firewalls, or a router/firewall and an end system

# Lecture 12 – Network Security (3)

- **Roadmap**

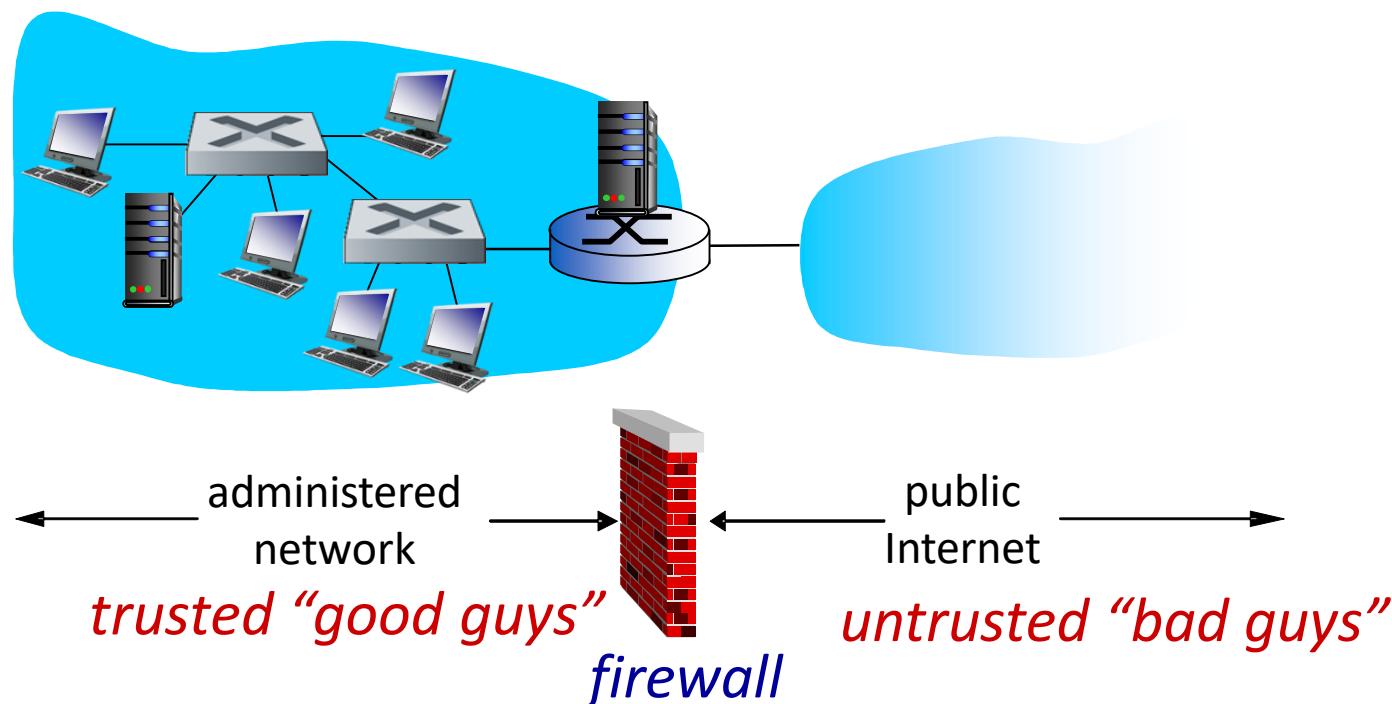
1. Network layer security: IPsec
2. Operational security: firewalls and IDS



# Firewalls

*firewall*

isolates organization's internal net from larger Internet,  
allowing some packets to pass, blocking others



# Firewalls: why

prevent denial of service attacks:

- SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections

prevent illegal modification/access of internal data

- e.g., attacker replaces CIA’s homepage with something else

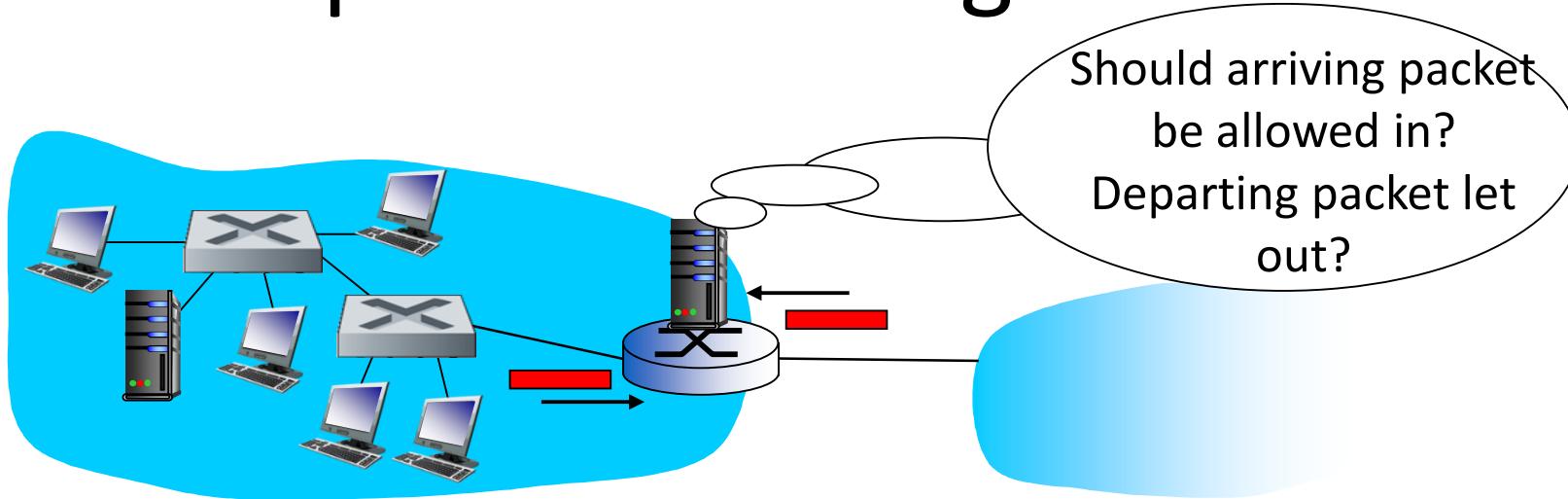
allow only authorized access to inside network

- set of authenticated users/hosts

three types of firewalls:

- stateless packet filters
- stateful packet filters
- application gateways

# Stateless packet filtering



- internal network connected to Internet via *router firewall*
- router *filters packet-by-packet*, decision to forward/drop packet based on:
  - source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits

# Stateless packet filtering: example

- *example 1:* block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
  - *result:* all incoming, outgoing UDP flows and telnet connections are blocked
- *example 2:* block inbound TCP segments with ACK=0.
  - *result:* prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.

# Stateless packet filtering: more examples

| <i>Policy</i>   | <i>Firewall Setting</i>  |
|---|--|
| No outside Web access.  | Drop all outgoing packets to any IP address, port 80                         |
| No incoming TCP connections, except those for institution's public Web server only. | Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80  |
| Prevent Web-radios from eating up the available bandwidth.                          | Drop all incoming UDP packets - except DNS and router broadcasts.            |
| Prevent your network from being used for a smurf DoS attack.                        | Drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255). |
| Prevent your network from being tracerouted   | Drop all outgoing ICMP TTL expired traffic                                   |

# Access Control Lists

**ACL:** table of rules, applied top to bottom to incoming packets:  
(action, condition) pairs: looks like OpenFlow forwarding table!

| action | source address          | dest address            | protocol | source port | dest port | flag bit |
|--------|-------------------------|-------------------------|----------|-------------|-----------|----------|
| allow  | 222.22/16               | outside of<br>222.22/16 | TCP      | > 1023      | 80        | any      |
| allow  | outside of<br>222.22/16 | 222.22/16               | TCP      | 80          | > 1023    | ACK      |
| allow  | 222.22/16               | outside of<br>222.22/16 | UDP      | > 1023      | 53        | ---      |
| allow  | outside of<br>222.22/16 | 222.22/16               | UDP      | 53          | > 1023    | ----     |
| deny   | all                     | all                     | all      | all         | all       | all      |

# Stateful packet filtering

- *stateless packet filter*: heavy handed tool
  - It may admit packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

| action | source address          | dest address | protocol | source port | dest port | flag bit |
|--------|-------------------------|--------------|----------|-------------|-----------|----------|
| allow  | outside of<br>222.22/16 | 222.22/16    | TCP      | 80          | > 1023    | ACK      |

- *stateful packet filter*: track status of every TCP connection
  - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
  - timeout inactive connections at firewall: no longer admit packets

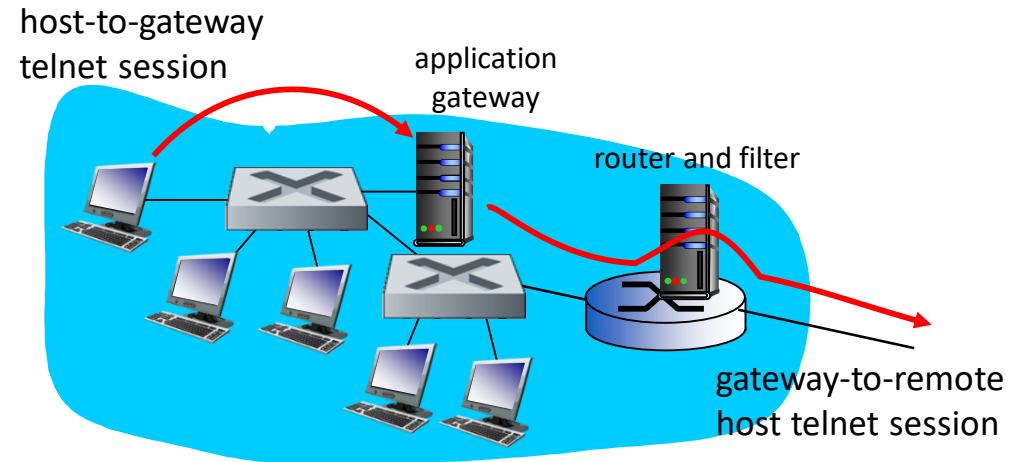
# Stateful packet filtering

ACL augmented to indicate need to check connection state table before admitting packet

| action | source address       | dest address         | proto | source port | dest port | flag bit | check connexion |
|--------|----------------------|----------------------|-------|-------------|-----------|----------|-----------------|
| allow  | 222.22/16            | outside of 222.22/16 | TCP   | > 1023      | 80        | any      |                 |
| allow  | outside of 222.22/16 | 222.22/16            | TCP   | 80          | > 1023    | ACK      | X               |
| allow  | 222.22/16            | outside of 222.22/16 | UDP   | > 1023      | 53        | ---      |                 |
| allow  | outside of 222.22/16 | 222.22/16            | UDP   | 53          | > 1023    | ----     | X               |
| deny   | all                  | all                  | all   | all         | all       | all      |                 |

# Application gateways

- filter packets on application data as well as on IP/TCP/UDP fields.
- *example:* allow select internal users to telnet outside



1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections.
3. router filter blocks all telnet connections not originating from gateway.

# Limitations of firewalls, gateways

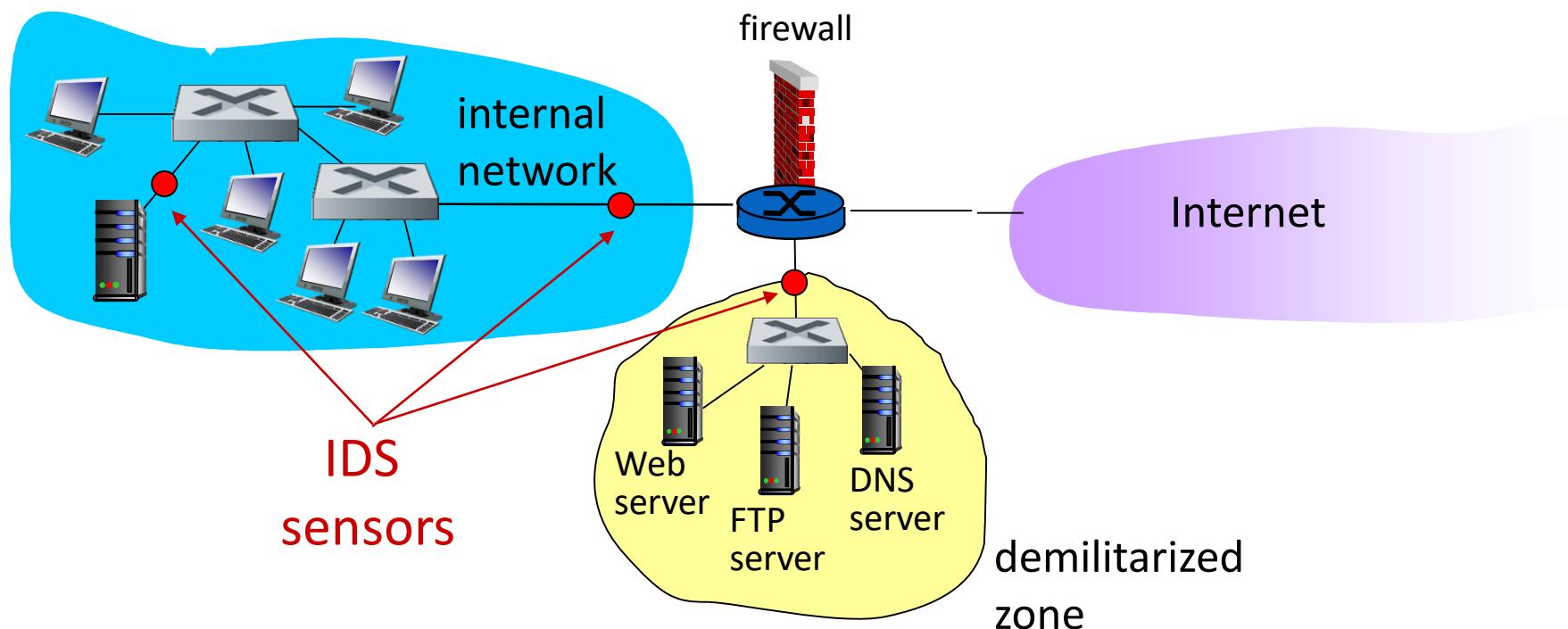
- *IP spoofing*: router can't know if data "really" comes from claimed source
- if multiple app's. need special treatment, each has own application gateway
- client software must know how to contact gateway.
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP
- *tradeoff*: degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks

# Intrusion detection systems

- **packet filtering:**
  - operates on TCP/IP headers only
  - no correlation check among sessions
- ***IDS: intrusion detection system***
  - *deep packet inspection*: look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
  - **examine correlation** among multiple packets
    - port scanning
    - network mapping
    - DoS attack

# Intrusion detection systems

multiple IDSs: different types of checking at different locations



# Network Security (summary)

**basic techniques.....**

- cryptography (symmetric and public)
- message integrity
- end-point authentication

**.... used in many different security scenarios**

- secure email
- secure transport (SSL)
- IP sec
- 802.11

**operational security: firewalls and IDS**

**Please do a survey on LMO.**

# General Feedback for In-class Test 2

- Each code should be tested on the VM of the Lab room (instead of the personal computer). That's why it is called **In-class Test!**
- General Issues
  - Wrong submission! (Everyone should take responsibility of the submitted code being the expected one)
  - Wrong IP addresses. (IP addresses should be checked on the created VM by corresponding Linux commands instead of looking at the code!)
  - Mistakenly using Controller. (Read the lab content and the topology diagram in the test specification)
- If anyone would like to appeal
  1. Go to the corresponding lab session this week. Ensure to go to the correct **lab group you belong to**, otherwise we won't accept the appeal.
  2. Let the TA download your code from LMO, test it on the VM of the Lab room, take a snapshot of the result and send it to the module leader.
  3. If you still don't agree the result from the VM, you can go to EE214 during the corresponding lab session you pertain to.
- Please do MQ.

# Good Luck!



# Thanks.

- **Addresses**
  - Email: [Wenjun.Fan@xjtu.edu.cn](mailto:Wenjun.Fan@xjtu.edu.cn)
  - Office: EE 214
- **Office hours**
  - Monday: 12:00 – 13:00
  - Tuesday: 12:00 – 13:00