# Chapter 8
# Network Security



## What is network security?

Confidentiality: only sender, intended receiver should "understand" message contents
- sender encrypts message
- receiver decrypts message

Authentication: sender, receiver want to confirm identity of each other

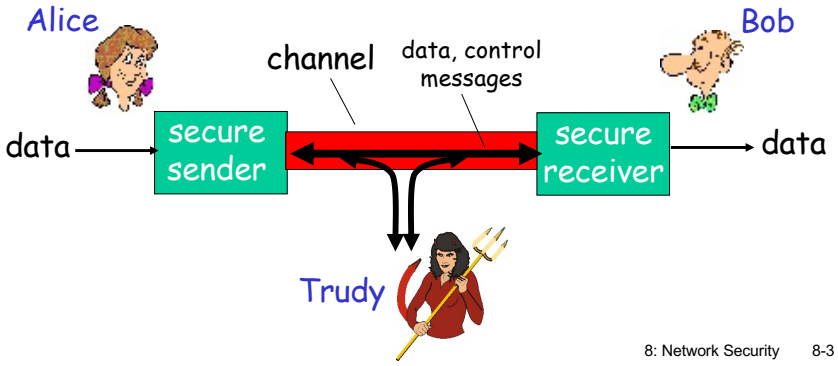
Message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

Access and availability: services must be accessible and available to users



## Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate "securely"
- Trudy (intruder) may intercept, delete, add messages






## Who might Bob, Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?

# There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

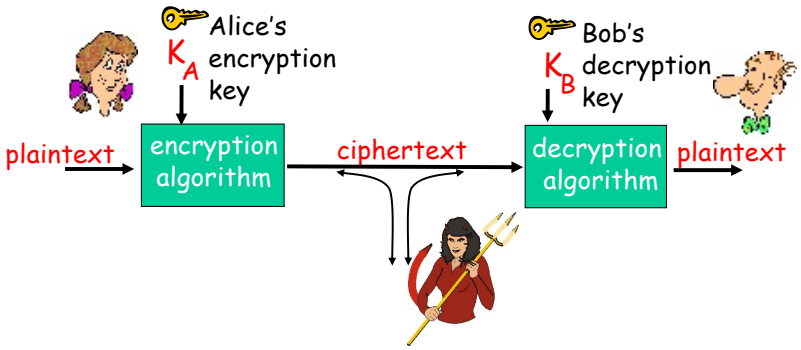
A: a lot!

- *eavesdrop:* intercept messages
- actively *insert* messages into connection
- *impersonation:* can fake (spoof) source address in packet (or any field in packet)
- *hijacking:* "take over" ongoing connection by removing sender or receiver, inserting himself in place
- *denial of service*: prevent service from being used by others (e.g., by overloading resources)

*more on this later ……*



# The language of cryptography

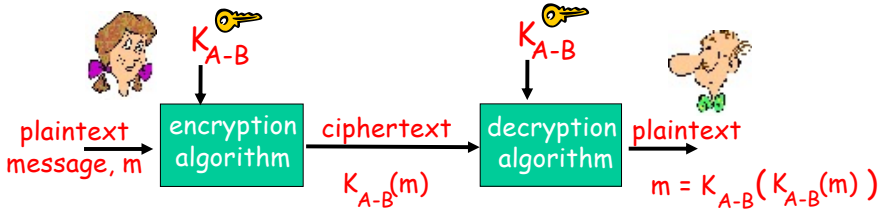



symmetric key crypto: sender, receiver keys *identical*

public-key crypto: encryption key *public*, decryption key *secret* (private)



# Symmetric key cryptography




symmetric key crypto: Bob and Alice share know same (symmetric) key: $K_{A-B}$

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher
- Q: how do Bob and Alice agree on key value?



# Symmetric key cryptography

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

```
plaintext:   abcdefghijklmnopqrstuvwxyz
ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

```
Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

*Encryption key:* mapping from set of 26 letters to set of 26 letters

# Symmetric key cryptography

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

```
plaintext:   abcdefghijklmnopqrstuvwxyz
             |                       |
ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

```
Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Q: How hard to break this simple cipher?:

- brute force (how hard?)
- other?



# A more sophisticated encryption approach

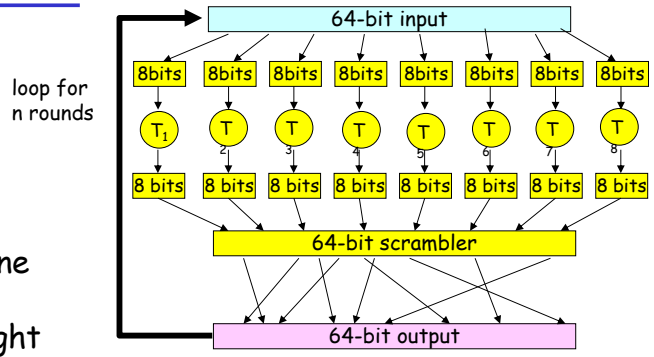
- n substitution ciphers, $M_1,M_2,\ldots,M_n$
- cycling pattern:
  - e.g., n=4: $M_1,M_3,M_4,M_3,M_2$; $M_1,M_3,M_4,M_3,M_2$; ..
- for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  - dog: d from $M_1$, o from $M_3$, g from $M_4$

*Encryption key:* n substitution ciphers, and cyclic pattern

- key need not be just n-bit pattern



# Block Cipher

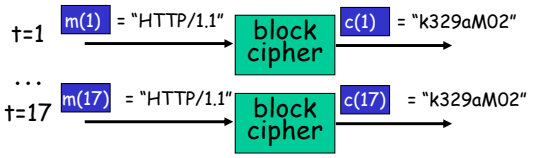



- one pass through: one input bit affects eight output bits
- multiple passes: each input bit afects all output bits
- block ciphers: DES, 3DES, AES

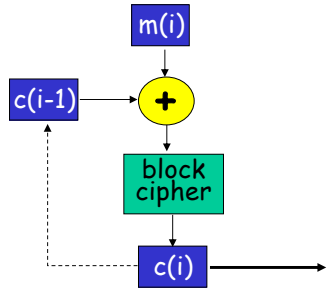


# Cipher Block Chaining

- cipher block: if input block repeated, will produce same cipher text:




- *cipher block chaining:* XOR ith input block, m(i), with previous block of cipher text, c(i-1)
  - c(0) transmitted to receiver in clear
  - what happens in "HTTP/1.1" scenario from above?



# Symmetric key crypto: DES

## DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase ("Strong cryptography makes the world a safer place") decrypted (brute force) in 4 months
  - no known "backdoor" decryption approach
- making DES more secure:
  - use three keys sequentially (3-DES) on each datum
  - use cipher-block chaining



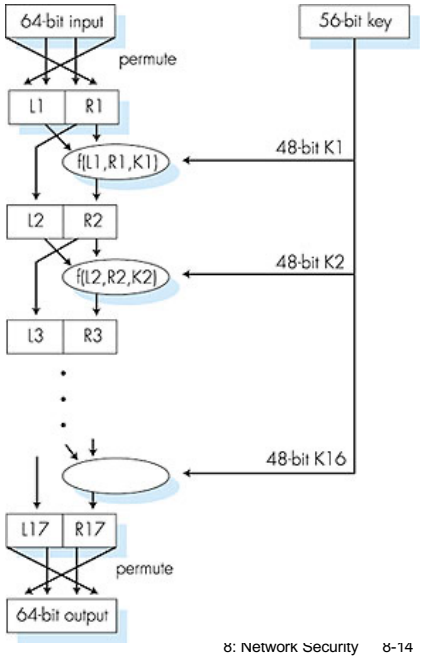
# Symmetric key crypto: DES

**DES operation**

initial permutation

16 identical "rounds" of function application, each using different 48 bits of key

final permutation






# AES: Advanced Encryption Standard

- new (Nov. 2001) symmetric-key NIST standard, replacing DES
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES



# Public key cryptography

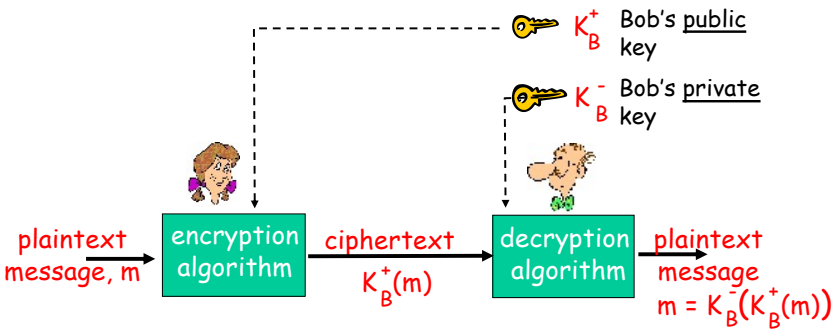
*symmetric* key crypto

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

*public* key cryptography

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver

# Public key cryptography






# Public key encryption algorithms

Requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that
   $K_B^-(K_B^+(m)) = m$
2. given public key $K_B^+$, it should be impossible to compute private key $K_B^-$

RSA: Rivest, Shamir, Adleman algorithm



# Prerequisite: modular arithmetic

- x mod n = remainder of x when divide by n
- facts:
  - [(a mod n) + (b mod n)] mod n = (a+b) mod n
  - [(a mod n) - (b mod n)] mod n = (a-b) mod n
  - [(a mod n) * (b mod n)] mod n = (a*b) mod n
- thus
  $(a \bmod n)^d \bmod n = a^d \bmod n$
- example: x=14, n=10, d=2:
  $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$
  $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$



# RSA: getting ready

- message: just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number

*example:*

- m= 10010001 . This message is uniquely represented by the decimal number 145.
- to encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext).

# RSA: Choosing keys

1. Choose two large prime numbers *p, q.* (e.g., 1024 bits each)
2. Compute $n = pq$, $z = (p-1)(q-1)$
3. Choose $e$ (with $e<n$) that has no common factors with z. (*e, z* are "relatively prime").
4. Choose $d$ such that $ed-1$ is exactly divisible by *z*. (in other words: $ed \bmod z = 1$ ).
5. *Public* key is $(n,e)$. *Private* key is $(n,d)$.

$K_B^+$ $K_B^-$



# RSA: Encryption, decryption

0. Given $(n,e)$ and $(n,d)$ as computed above
1. To encrypt bit pattern, *m*, compute $c = m^e \bmod n$ (i.e., remainder when $m^e$ is divided by *n*)
2. To decrypt received bit pattern, *c*, compute $m = c^d \bmod n$ (i.e., remainder when $c^d$ is divided by *n*)

Magic happens! $m = \underbrace{(m^e \bmod n)}_{c}{}^d \bmod n$



# RSA example:

Bob chooses $p=5$, $q=7$. Then $n=35$, $z=24$.
$e=5$ (so *e, z* relatively prime).
$d=29$ (so $ed-1$ exactly divisible by z.

encrypt:

| letter | m | $m^e$ | $c = m^e \bmod n$ |
|---|---|---|---|
| l | 12 | 1524832 | 17 |

decrypt:

| c | $c^d$ | $m = c^d \bmod n$ | letter |
|---|---|---|---|
| 17 | 481968572106750915091411825223071697 | 12 | l |



# RSA: Why is that $m = (m^e \bmod n)^d \bmod n$

Useful number theory result: If *p,q* prime and $n = pq$, then: $x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$
$$= m^{ed \bmod (p-1)(q-1)} \bmod n$$

(using number theory result above)

$$= m^1 \bmod n$$

(since we chose $ed$ to be divisible by $(p-1)(q-1)$ with remainder 1 )

$$= m$$

## RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

*Result is the same!*



## Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$ ?

follows directly from modular arithmetic:

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$
$$= m^{de} \bmod n$$
$$= (m^d \bmod n)^e \bmod n$$



## Why is RSA secure?

- suppose you know Bob's public key (n,e). How hard is it to determine d?
- essentially need to find factors of n without knowing the two factors p and q
  - fact: factoring a big number is hard



## RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

*session key, $K_S$*

- Bob and Alice use RSA to exchange a symmetric key $K_S$
- once both have $K_S$, they use symmetric key cryptography

# Message Integrity

Bob receives msg from Alice, wants to ensure:

- message originally came from Alice
- message not changed since sent by Alice

## Cryptographic Hash:

- takes input m, produces fixed length value, H(m)
  - e.g., as in Internet checksum
- computationally infeasible to find two different messages, x, y such that H(x) = H(y)
  - equivalently: given m = H(x), (x unknown), can not determine x.
  - note: Internet checksum *fails* this requirement!



# Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- ✓ produces fixed length digest (16-bit sum) of message
- ✓ is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

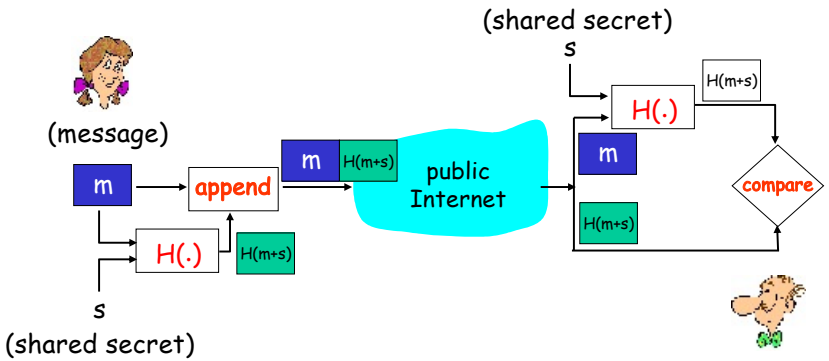
| message | ASCII format |
|---|---|
| I O U 1 | 49 4F 55 31 |
| 0 0 . 9 | 30 30 2E 39 |
| 9 B O B | 39 42 4F 42 |
| | B2 C1 D2 AC |

different messages but identical checksums!

| message | ASCII format |
|---|---|
| I O U 9 | 49 4F 55 39 |
| 0 0 . 1 | 30 30 2E 31 |
| 9 B O B | 39 42 4F 42 |
| | B2 C1 D2 AC |



# Message Authentication Code






# MACs in practice

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit MAC in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x
    - recent (2005) attacks on MD5
- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit MAC

# Digital Signatures

cryptographic technique analogous to hand-written signatures.

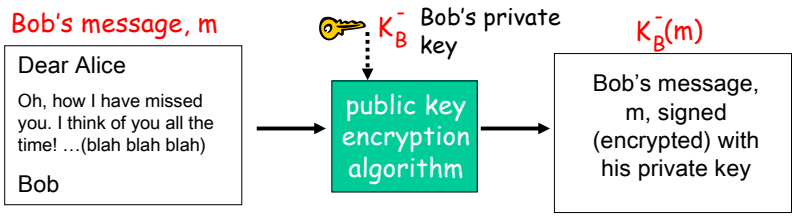
- sender (Bob) digitally signs document, establishing he is document owner/creator.
- verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document



# Digital Signatures

simple digital signature for message m:

- Bob "signs" m by encrypting with his private key $K_B^-$, creating "signed" message, $K_B^-(m)$






# Digital Signatures (more)

- suppose Alice receives msg m, digital signature $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key $K_B^+$ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- if $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

Alice thus verifies that:

- ✓ Bob signed m.
- ✓ No one else signed m.
- ✓ Bob signed m and not m'.

non-repudiation:

- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.

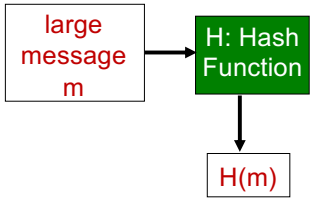


# Message digests

computationally expensive to public-key-encrypt long messages

*goal:* fixed-length, easy-to-compute digital "fingerprint"

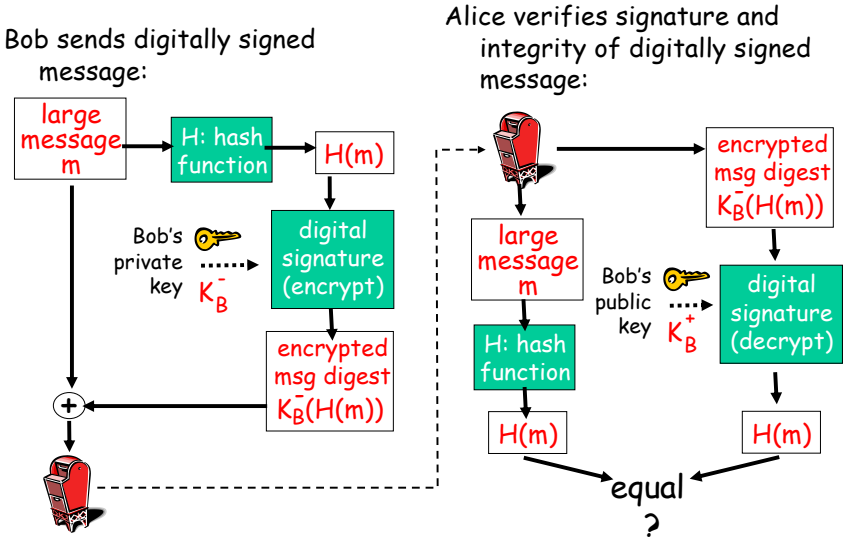
- apply hash function H to *m*, get fixed size message digest, *H(m)*.




Hash function properties:

- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest x, computationally infeasible to find m such that x = H(m)

# Digital signature = signed MAC

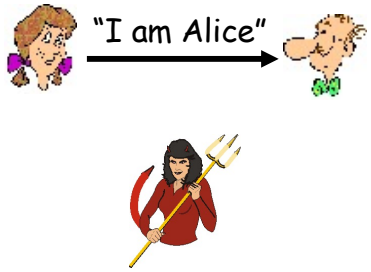





# Authentication

Goal: Bob wants Alice to "prove" her identity to him

Protocol ap1.0: Alice says "I am Alice"



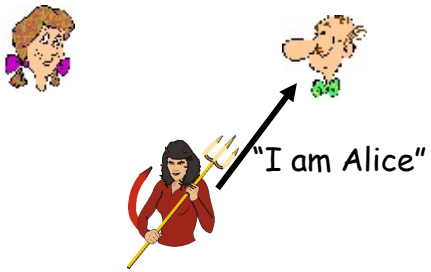

Failure scenario??



# Authentication

Goal: Bob wants Alice to "prove" her identity to him

Protocol ap1.0: Alice says "I am Alice"




in a network, Bob can not "see" Alice, so Trudy simply declares herself to be Alice



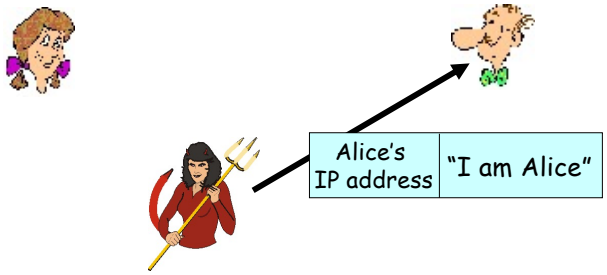
# Authentication: another try

Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address


Failure scenario??

# Authentication: another try

Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address

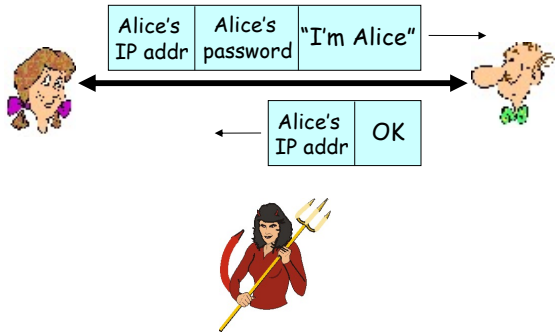



Trudy can create a packet "spoofing" Alice's address



# Authentication: another try

Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.

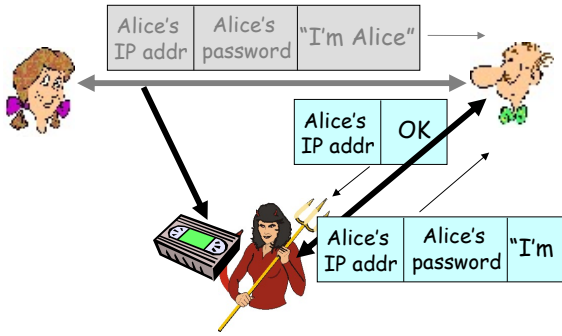



Failure scenario??



# Authentication: another try

Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.

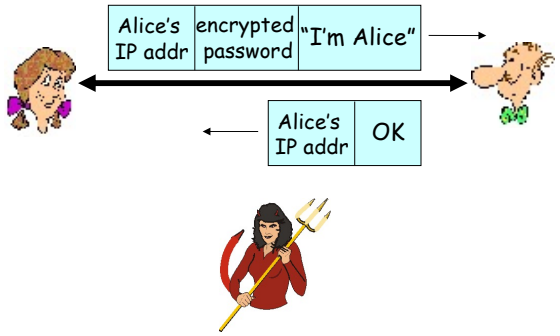



*playback attack:* Trudy records Alice's packet and later plays it back to Bob



# Authentication: yet another try

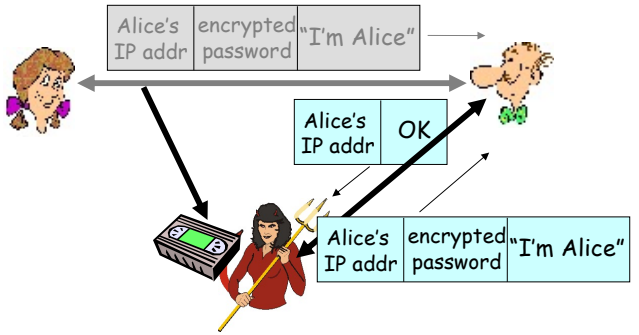
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.




Failure scenario??

# Authentication: another try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.




record and playback still works!

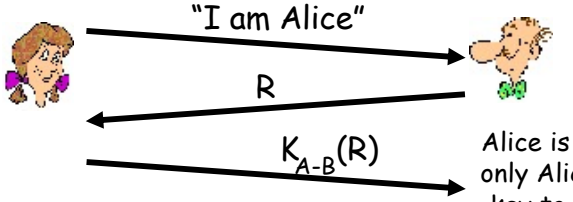


# Authentication: yet another try

Goal: avoid playback attack

Nonce: number (R) used only *once -in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key




Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

Failures, drawbacks?

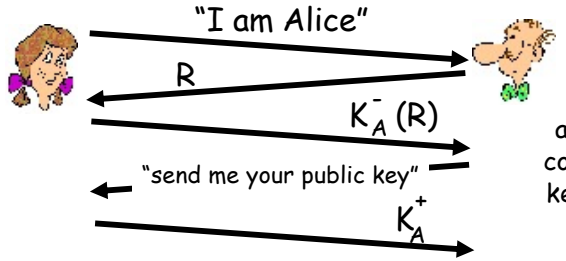


# Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography

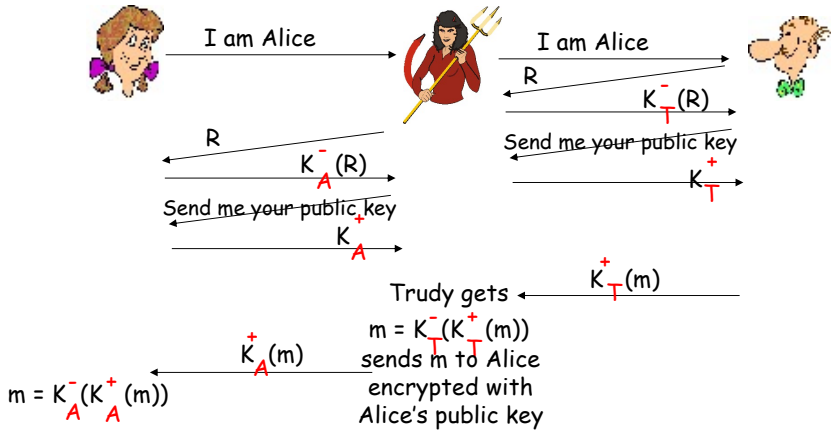



Bob computes $K_A^+(K_A^-(R)) = R$ and knows only Alice could have the private key, that encrypted R such that $K_A^+(K_A^-(R)) = R$



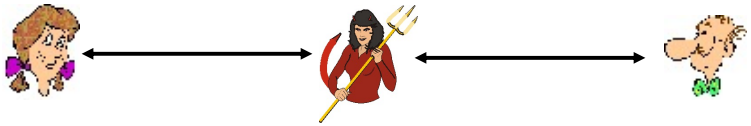
# ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



# ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation)
- problem is that Trudy receives all messages as well!



# Public Key Certification

public key problem:

- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she *know* it is Bob's public key, not Trudy's?

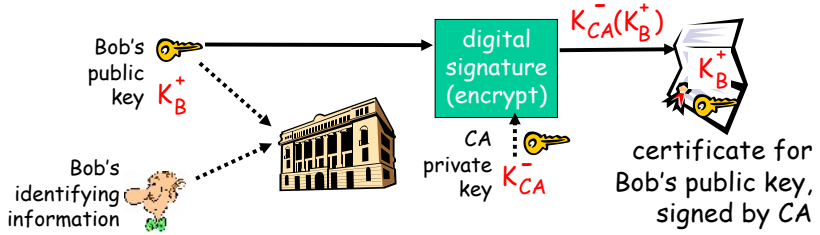
solution:

- trusted certification authority (CA)



# Certification Authorities

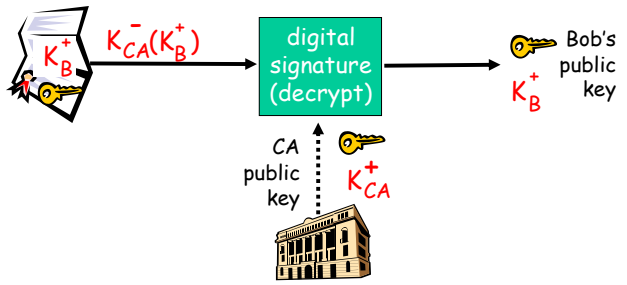
- Certification Authority (CA): binds public key to particular entity, E.
- E registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA: CA says "This is E's public key."





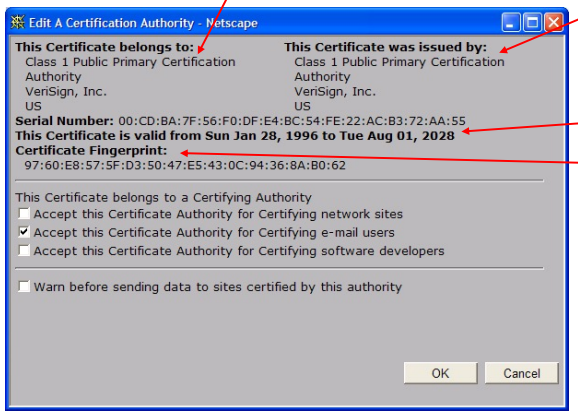

# Certification Authorities

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



# A certificate contains:

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)

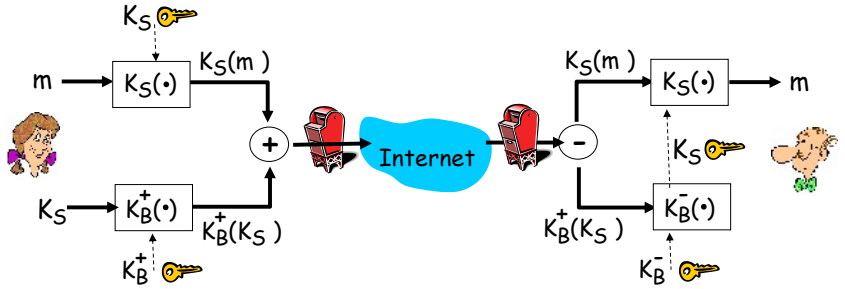



- info about certificate issuer
- valid dates
- digital signature by issuer



# Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.



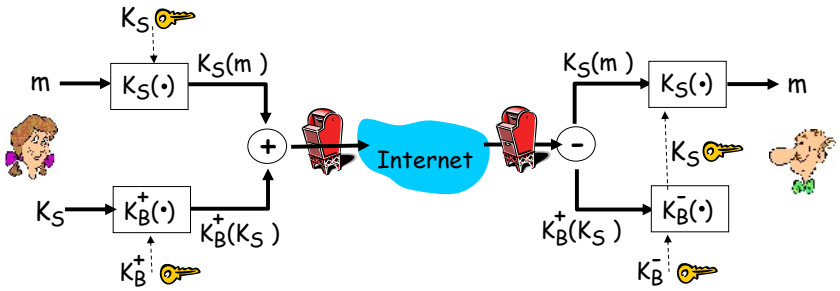

Alice:

- generates random *symmetric* private key, $K_S$.
- encrypts message with $K_S$ (for efficiency)
- also encrypts $K_S$ with Bob's public key.
- sends both $K_S(m)$ and $K_B(K_S)$ to Bob.



# Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.



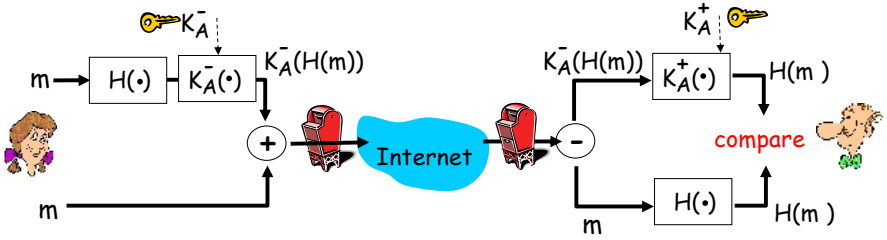

Bob:

- uses his private key to decrypt and recover $K_S$
- uses $K_S$ to decrypt $K_S(m)$ to recover m



# Secure e-mail (continued)

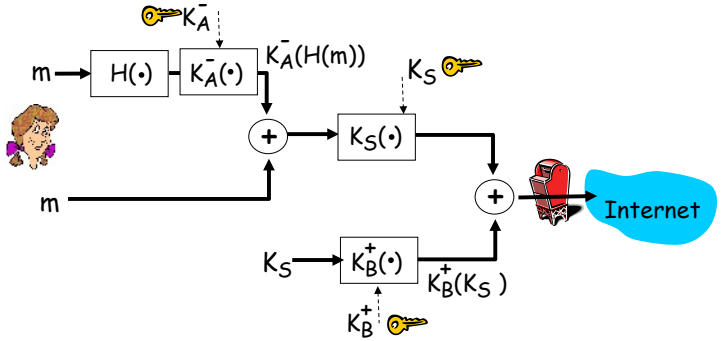
• Alice wants to provide sender authentication message integrity.




- Alice digitally signs message.
- sends both message (in the clear) and digital signature.

# Secure e-mail (continued)

- Alice wants to provide secrecy, sender authentication, message integrity.




Alice uses three keys: her private key, Bob's public key, newly created symmetric key



# Pretty good privacy (PGP)

- Internet e-mail encryption scheme, de-facto standard.
- uses symmetric key cryptography, public key cryptography, hash function, and digital signature as described.
- provides secrecy, sender authentication, integrity.
- inventor, Phil Zimmerman, was target of 3-year federal investigation.

A PGP signed message:

```
---BEGIN PGP SIGNED MESSAGE---
Hash: SHA1

Bob:My husband is out of town
   tonight.Passionately yours,
   Alice

---BEGIN PGP SIGNATURE---
Version: PGP 5.0
Charset: noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJ
   hFEvZP9t6n7G6m5Gw2
---END PGP SIGNATURE---
```

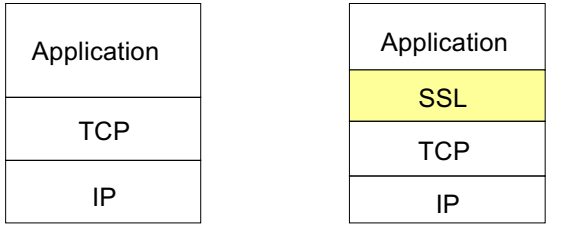


# SSL: Secure Sockets Layer

- widely deployed security protocol
  - supported by almost all browsers, web servers
  - https
  - billions $/year over SSL
- mechanisms: [Woo 1994], implementation: Netscape
- variation -TLS: transport layer security, RFC 2246
- provides
  - *confidentiality*
  - *integrity*
  - *authentication*
- original goals:
  - Web e-commerce transactions
  - encryption (especially credit-card numbers)
  - Web-server authentication
  - optional client authentication
  - minimum hassle in doing business with new merchant
- available to all TCP applications
  - secure socket interface



# SSL and TCP/IP




*normal application* *application with SSL*

- SSL provides application programming interface (API) to applications
- C and Java SSL libraries/classes readily available

# Toy SSL: a simple secure channel

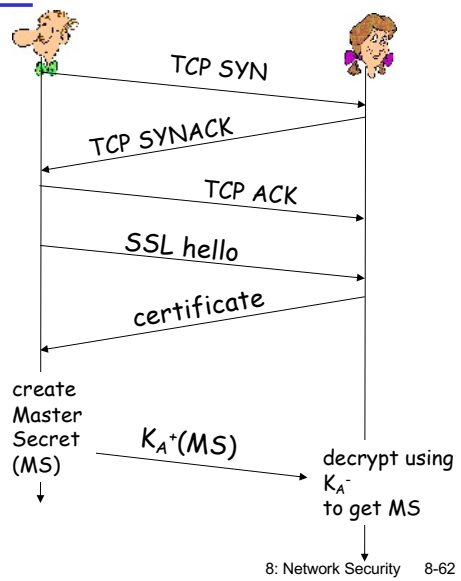
- *handshake:* Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret
- *key derivation:* Alice and Bob use shared secret to derive set of keys
- *data transfer:* data to be transferred is broken up into series of records
- *connection closure:* special messages to securely close connection



# SSL: three phases

*1. Handshake:*

- Bob establishes TCP connection to Alice
- authenticates Alice via CA signed certificate
- creates, encrypts (using Alice's public key), sends master secret key to Alice
  - nonce exchange not shown






# SSL: three phases

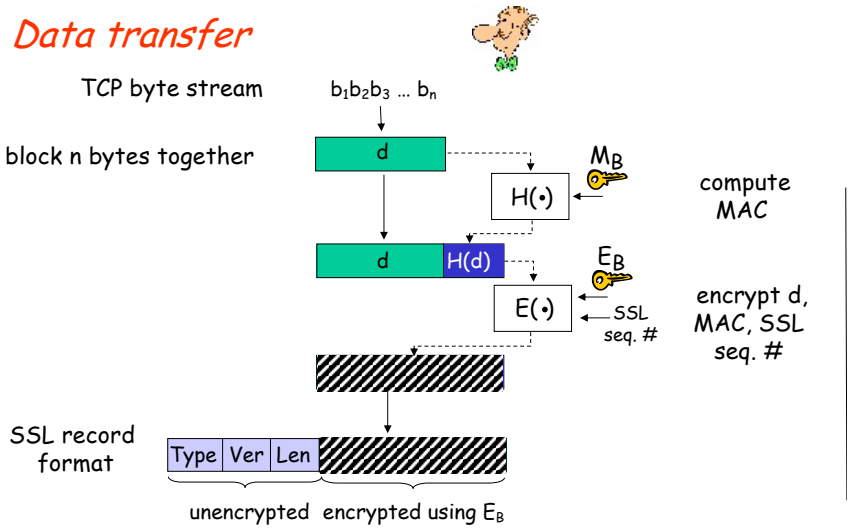
*2. Key Derivation:*

- Alice, Bob use shared secret (MS) to generate 4 keys:
  - $E_B$: Bob->Alice data encryption key
  - $E_A$: Alice->Bob data encryption key
  - $M_B$: Bob->Alice MAC key
  - $M_A$: Alice->Bob MAC key
- encryption and MAC algorithms negotiable between Bob, Alice
- why 4 keys?

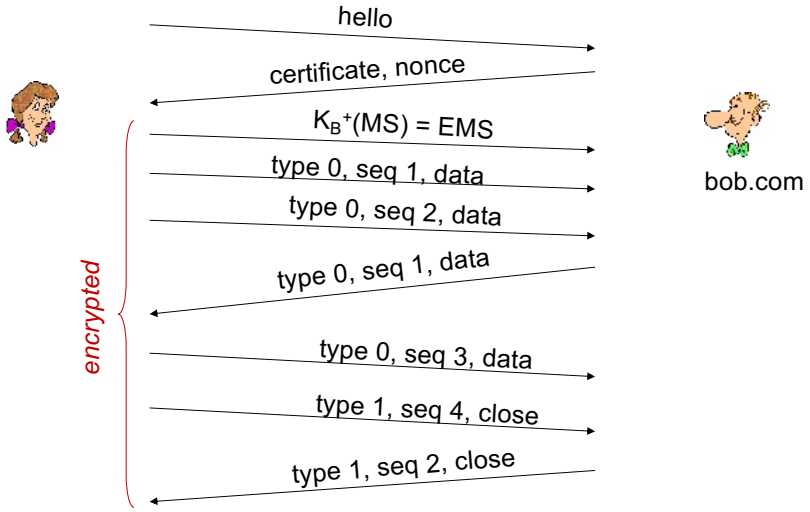


# SSL: three phases

*3. Data transfer*



# Toy SSL: summary






# Toy SSL isn't complete

- how long are fields?
- which encryption protocols?
- want negotiation?
  - allow client and server to support different encryption algorithms
  - allow client and server to choose together specific algorithm before data transfer



# 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:
  - web pages, e-mail, P2P file transfers, TCP SYN packets …
- "blanket coverage"

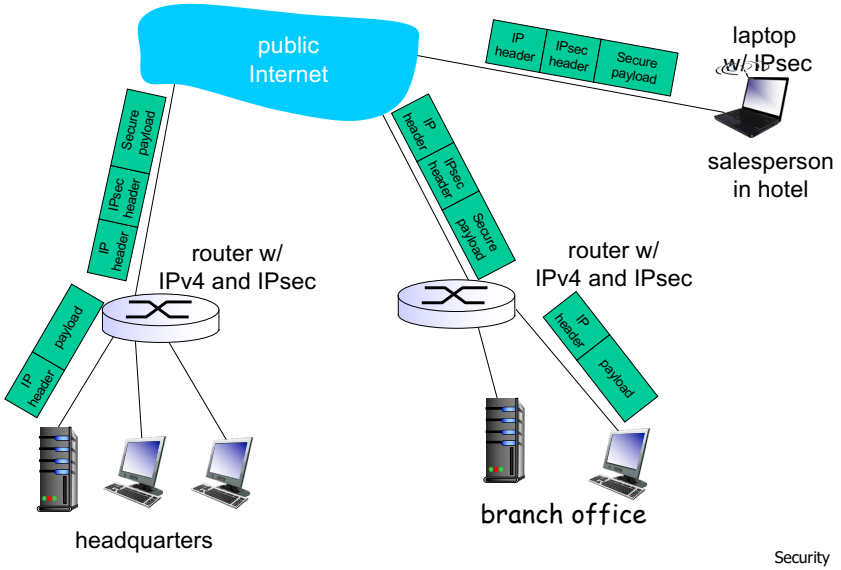


# Virtual Private Networks (VPNs)

*motivation:*

- institutions often want private networks for security.
  - 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

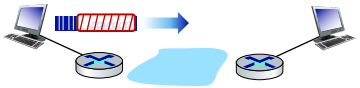
- data integrity
- origin authentication
- replay attack prevention
- confidentiality

- two protocols providing different service models:
  - AH
  - ESP

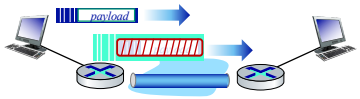


# IP Sec

- provides datagram-level encryption, authentication, integrity
  - for both user traffic and control traffic (e.g., BGP, DNS messages)
- two "modes":



transport mode:
- *only* datagram *payload* is encrypted, authenticated



tunnel mode:
- entire datagram is encrypted, authenticated
- encrypted datagram encapsulated in new datagram with new IP header, tunneled to destination



# Two IPsec protocols

- Authentication Header (AH) protocol
  - provides source authentication & data integrity but *not* confidentiality
- Encapsulation Security Protocol (ESP)
  - provides source authentication, data integrity, *and confidentiality*
  - more widely used than AH

# Four combinations are possible!

| | |
|---|---|
| Host mode with AH | Host mode with ESP |
| Tunnel mode with AH | Tunnel mode with ESP |

most common and most important

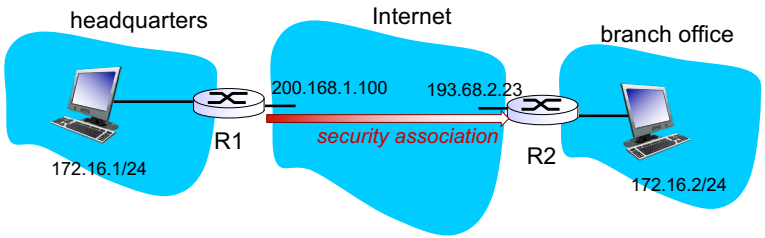


# Security associations (SAs)

- before sending data, "security association (SA)" established from sending to receiving entity
  - SAs are simplex: 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/ headquarters, branch office, and n traveling salespeople?



# 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

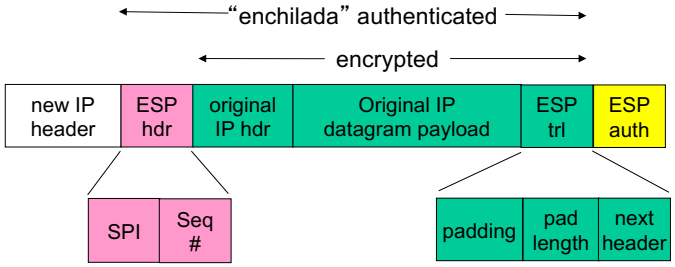


# Security Association Database (SAD)

- 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.

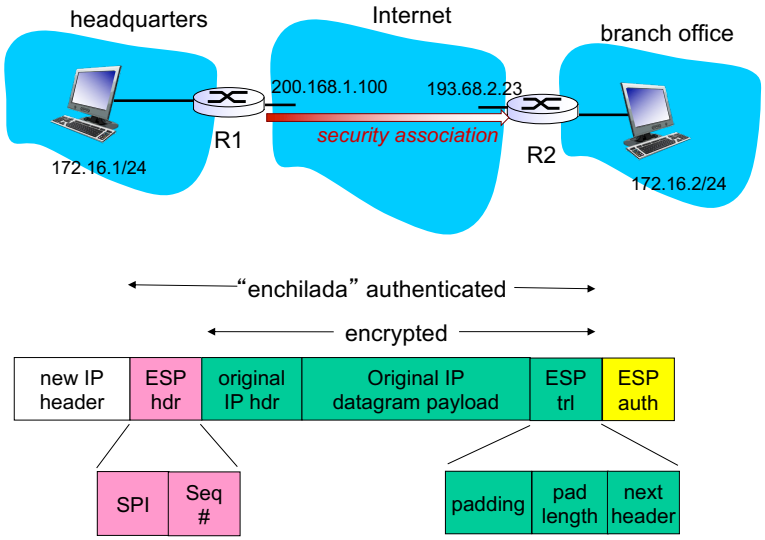
# IPsec datagram

focus on tunnel mode with ESP






# What happens?



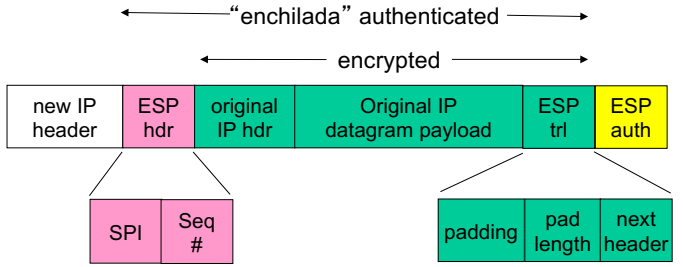



# R1: convert original datagram to IPsec datagram

- appends to back of original datagram (which 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
- 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)*

# IKE: PSK and PKI

- authentication (prove who you are) with either
  - pre-shared secret (PSK) or
  - with PKI (public/private keys and certificates).
- PSK: both sides start with secret
  - run IKE to authenticate each other and to generate IPsec SAs (one in each direction), including encryption, authentication keys
- PKI: both sides start with public/private key pair, certificate
  - run IKE to authenticate each other, obtain IPsec SAs (one in each direction).
  - similar with handshake in SSL.



# IPsec summary

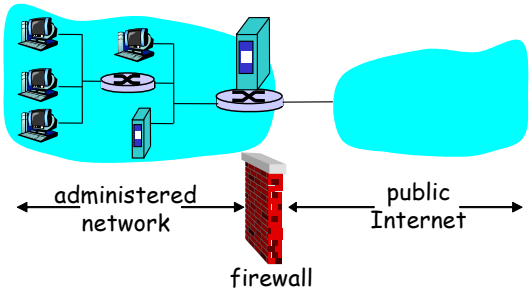
- IKE message exchange for algorithms, secret keys, SPI numbers
- either AH or ESP protocol (or both)
  - AH provides integrity, source authentication
  - ESP protocol (with AH) additionally provides encryption
- IPsec peers can be two end systems, two routers/firewalls, or a router/firewall and an end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



# 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.

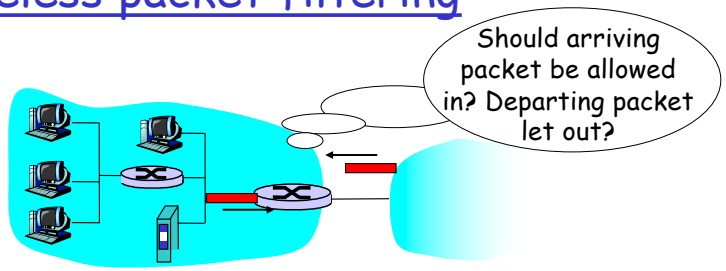
- e.g., attacker replaces CIA's homepage with something else

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



# Stateless packet filtering: more examples

| Policy | Firewall Setting |
|---|---|
| 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 (eg 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

| action | source address | dest address | protocol | source port | dest port | flag bit |
|---|---|---|---|---|---|---|
| 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 |
| allow | 222.22/16 | outside of 222.22/16 | UDP | > 1023 | 53 | --- |
| allow | outside of 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
  - admits 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 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): can determine whether incoming, outgoing packets "makes sense"
  - timeout inactive connections at firewall: no longer admit packets



# Stateful packet filtering

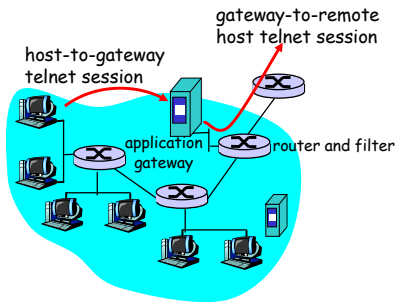
- ACL augmented to indicate need to check connection state table before admitting packet

| action | source address | dest address | proto | source port | dest port | flag bit | check conxion |
|---|---|---|---|---|---|---|---|
| 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

- filters 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 and gateways

- IP spoofing: router can't know if data "really" comes from claimed source
- if multiple app's. need special treatment, each has own app. 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.

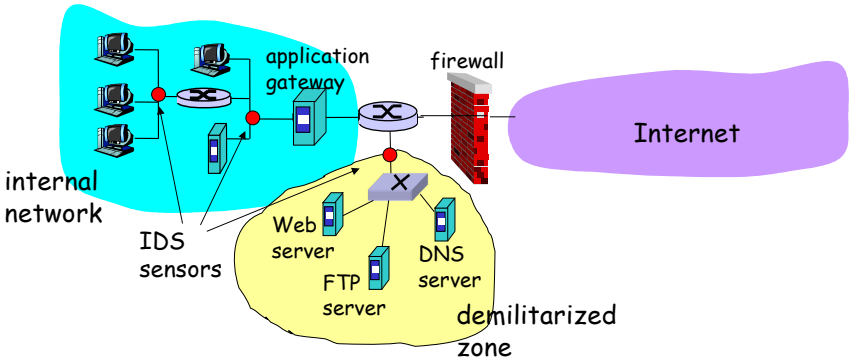
# Intrusion detection systems

- 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

Operational Security: firewalls and IDS