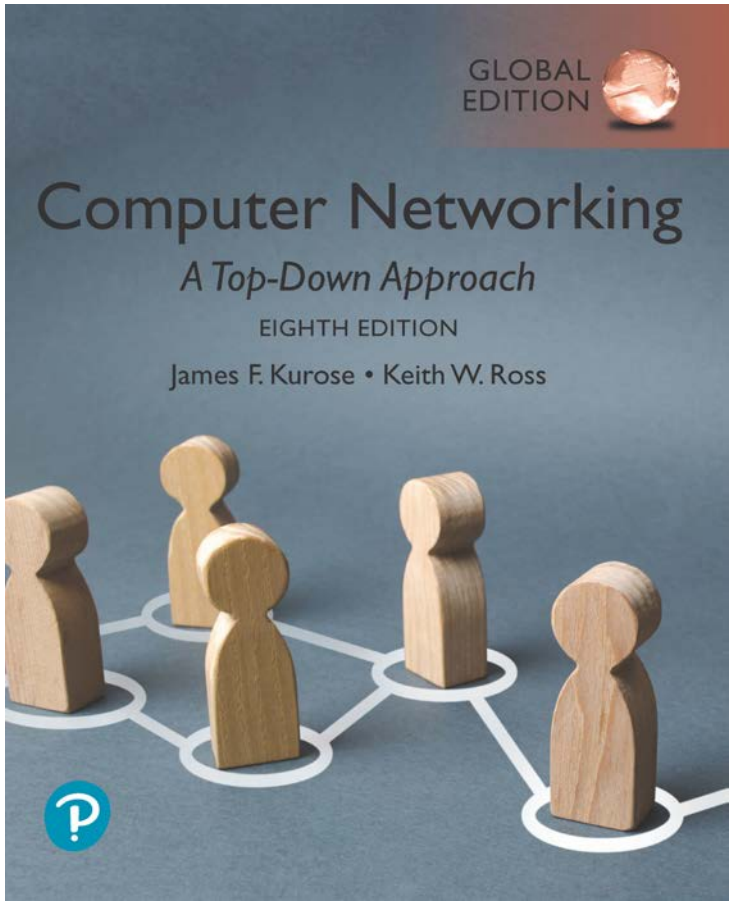
# Chapter 2
# Security in Computer Networks




8th Edition, Global Edition, Jim Kurose, Keith Ross
Copyright © 2022 Pearson Education Ltd

# Security: overview

## Chapter goals:

- understand principles of network security:
  - cryptography and its *many* uses beyond "confidentiality"
  - authentication
  - message integrity
- security in practice:
  - firewalls and intrusion detection systems
  - security in application, transport, network, link layers

# Chapter 2 outline

- What is network security?
- Principles of cryptography
- Message integrity, authentication
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

# What is network security?

confidentiality: only sender, intended receiver should "understand" message contents

- sender encrypts message
- receiver decrypts message

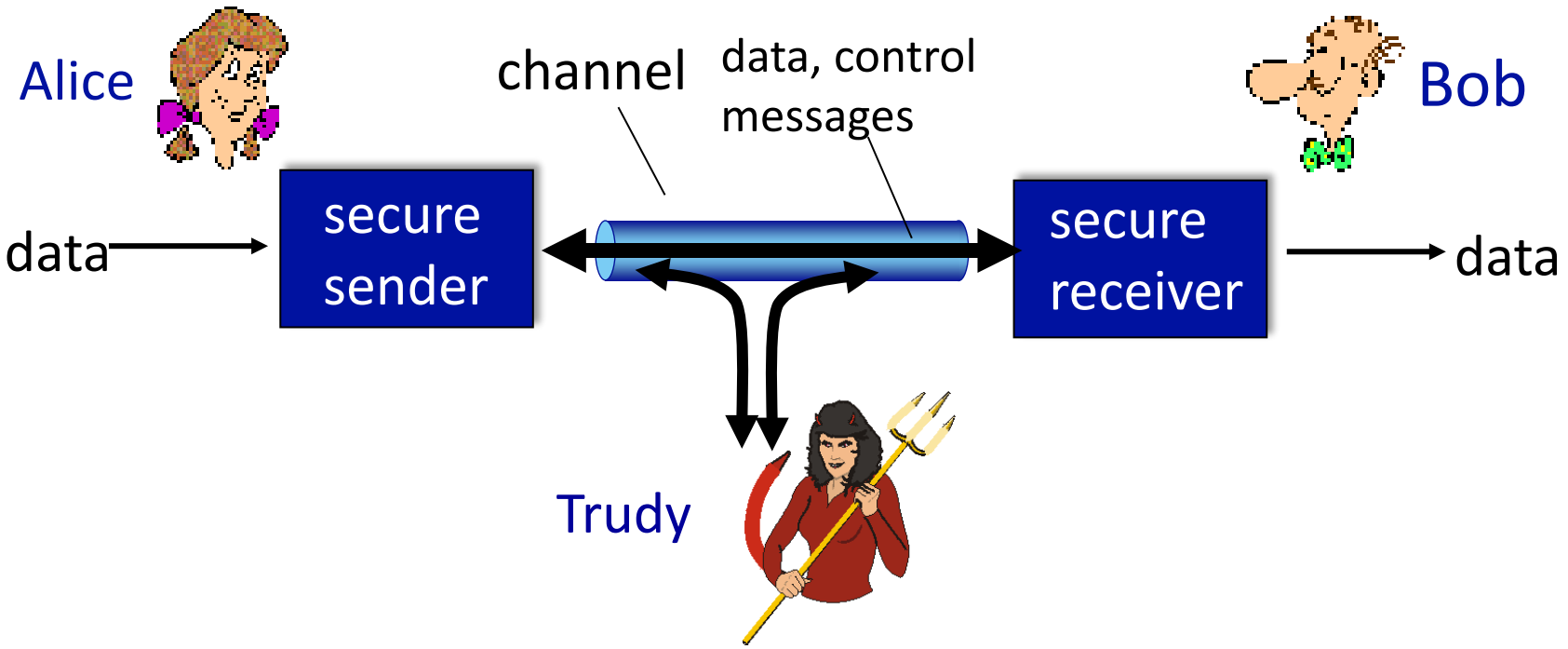
authentication: sender, receiver want to confirm identity of each other

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



# Friends and enemies: Alice, Bob, Trudy

Who might Bob and 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
- BGP routers exchanging routing table updates
- other examples?

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

*Q:* What can a "bad guy" do?

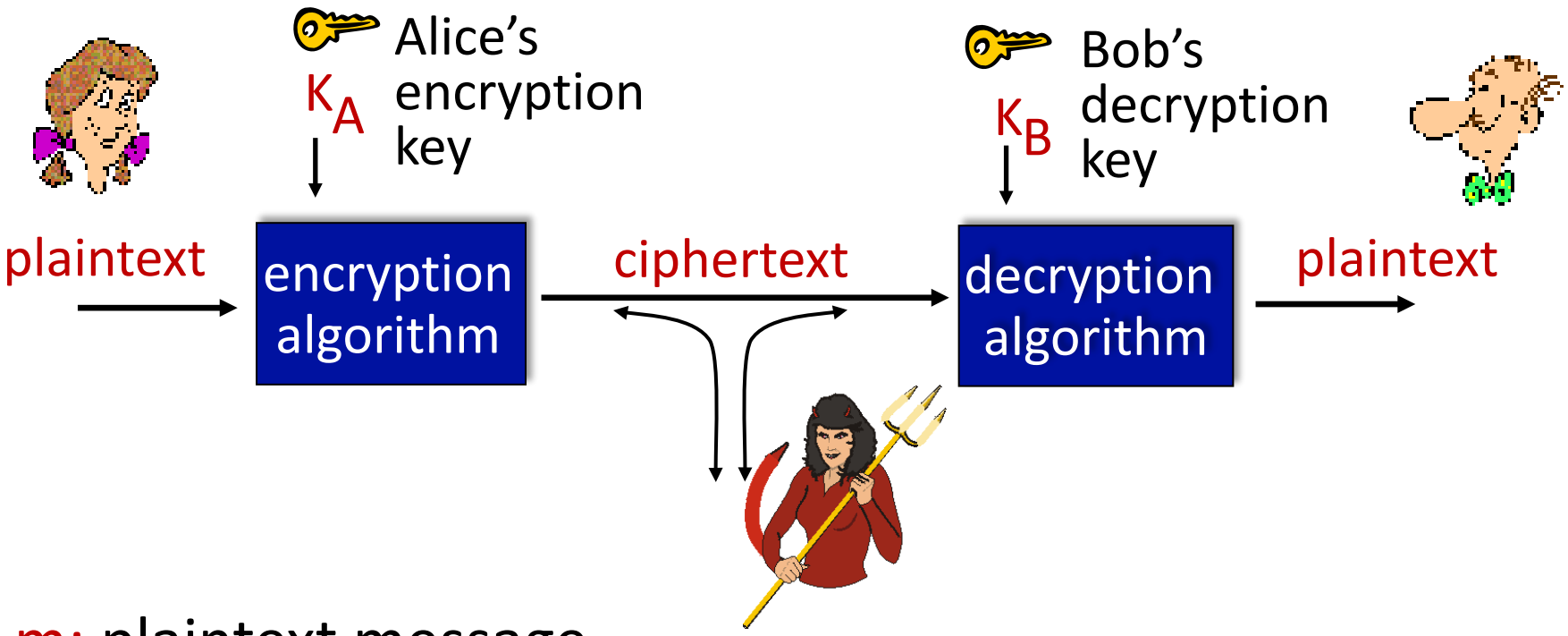
*A:* A lot!

- 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)

# Chapter 2 outline

- What is network security?
- Principles of cryptography
- Message integrity, authentication
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

# The language of cryptography




m: plaintext message

$K_A(m)$: ciphertext, encrypted with key $K_A$

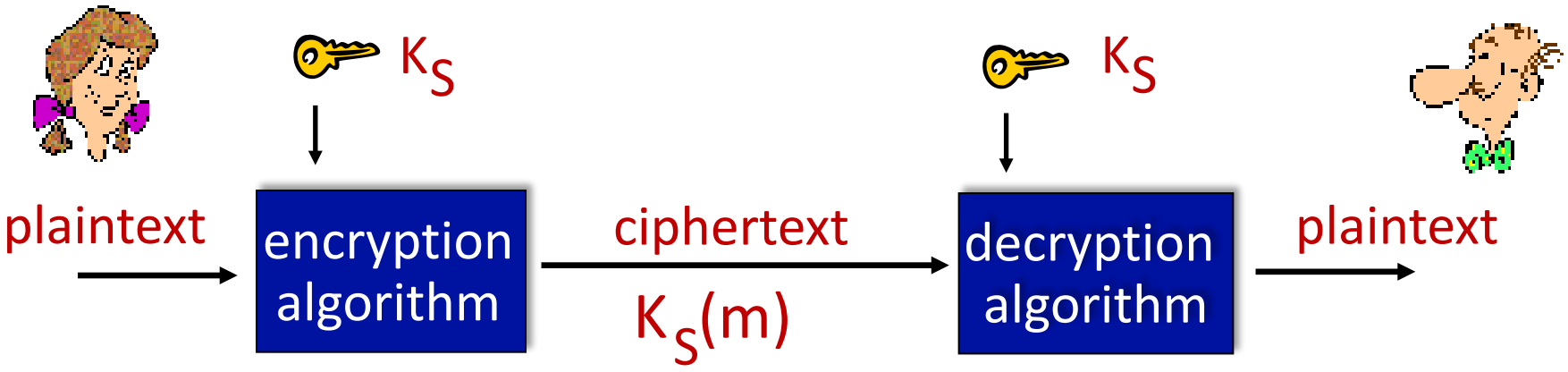
$m = K_B(K_A(m))$

# Breaking an encryption scheme

- cipher-text only attack:
  Trudy has ciphertext she can analyze
- two approaches:
  - brute force: search through all keys
  - statistical analysis

- known-plaintext attack:
  Trudy has plaintext corresponding to ciphertext
  - *e.g.*, in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,
- chosen-plaintext attack:
  Trudy can get ciphertext for chosen plaintext

# Symmetric key cryptography




symmetric key crypto: Bob and Alice share same (symmetric) key: K

- *e.g.,* key is knowing substitution pattern in mono alphabetic substitution cipher

*Q:* how do Bob and Alice agree on key value?

# Simple encryption scheme

*substitution cipher:* substituting one thing for another

- monoalphabetic cipher: substitute one letter for another


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

# 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

# Symmetric key crypto: DES

## DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- how secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase  decrypted (brute force) in less than a day
  - no known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

# AES: Advanced Encryption Standard

- symmetric-key NIST standard, replaced DES (Nov 2001)
- 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

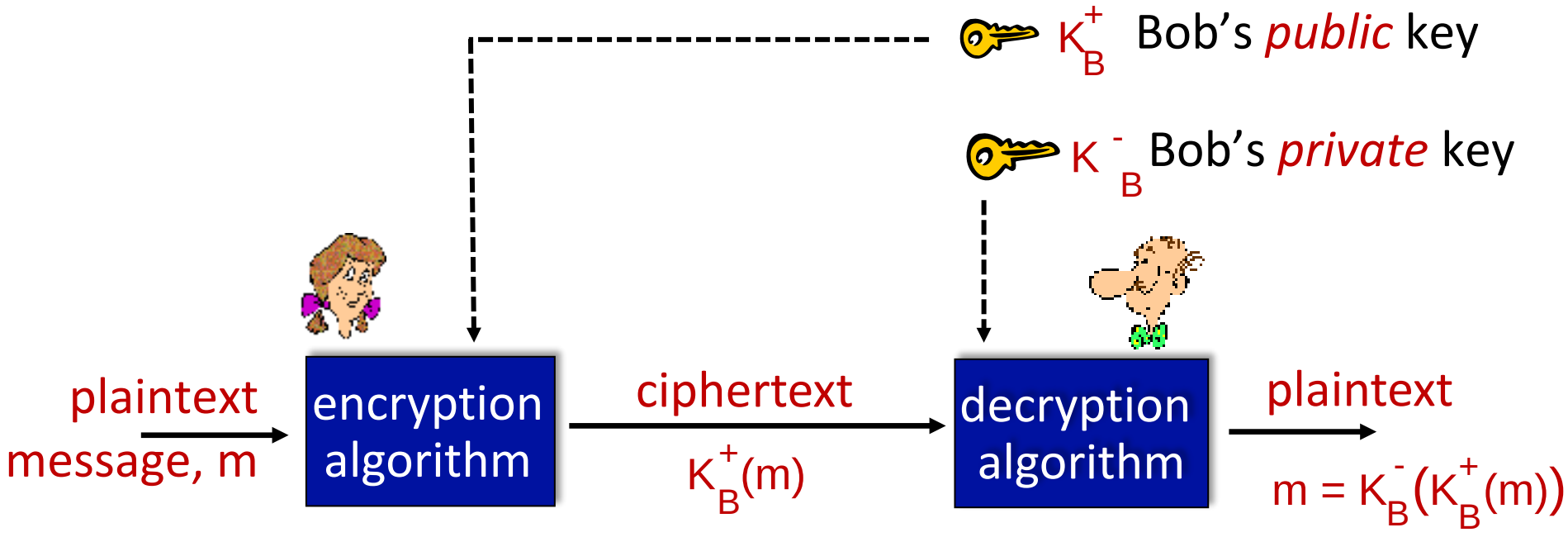
## symmetric key crypto:

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

## public key crypto

- *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




*Wow* - public key cryptography revolutionized 2000-year-old (previously only symmetric key) cryptography!

- similar ideas emerged at roughly same time, independently in US and UK (classified)

# 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, Adelson 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: Creating public/private key pair

1. choose two large prime numbers *p, q.* (e.g., 1024 bits each)

2. compute *n = pq, z = (p-1)(q-1)*

Public Exponent

3. choose *e (*with *e<n)* that has no common factors with z (*e, z* are "relatively prime").

Private Exponent

4. choose *d* such that *ed-1* is exactly divisible by *z*. (in other words: *ed* mod *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 message *m* (*<n*), compute

$$c = m^e \bmod n$$

2. to decrypt received bit pattern, *c*, compute

$$m = c^d \bmod n$$

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

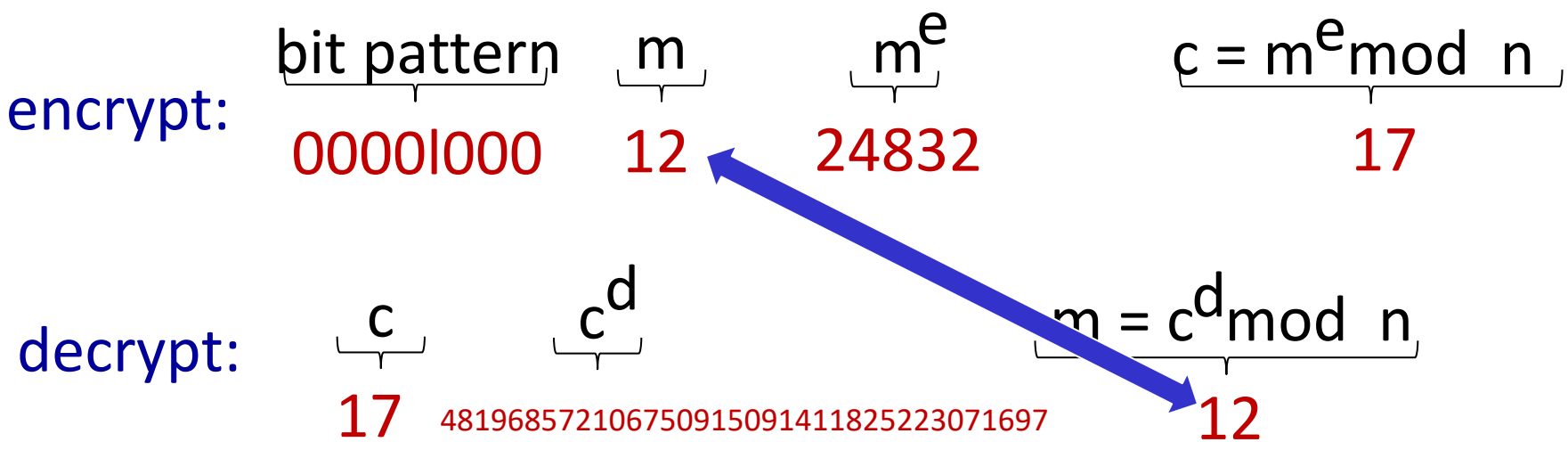
# RSA example:

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).

encrypting 8-bit messages.



# Why does RSA work?

- must show that $c^d \bmod n = m$, where $c = m^e \bmod n$
- fact: for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - where n= pq and z = (p-1)(q-1)
- thus,

$$c^d \bmod n = (m^e \bmod n)^d \bmod n$$
$$= m^{ed} \bmod n$$
$$= m^{(ed \bmod z)} \bmod n$$
$$= m^1 \bmod n$$
$$= m$$

# 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 session key $K_S$
- once both have $K_S$, they use symmetric key cryptography

# Chapter 2 outline

- What is network security?
- Principles of cryptography
- **Authentication**, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

# Authentication

Goal: Bob wants Alice to “prove” her identity to him

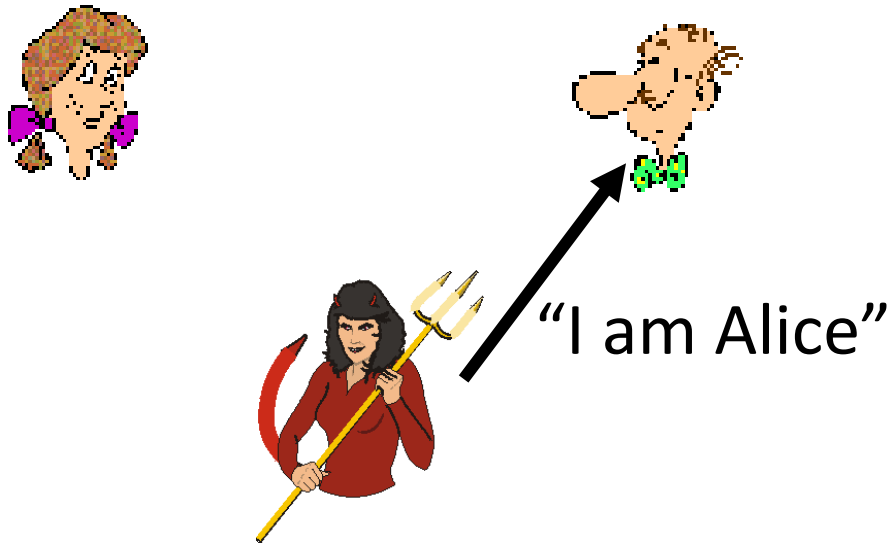
Protocol ap1.0: Alice says “I am Alice”


*failure scenario??*

# Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”

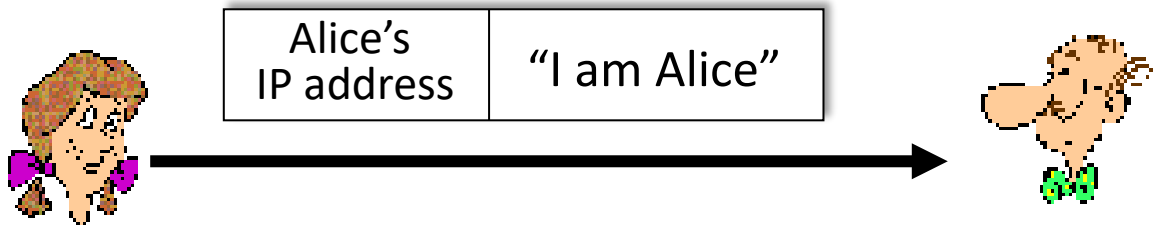



*in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice*

# Authentication: another try

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

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

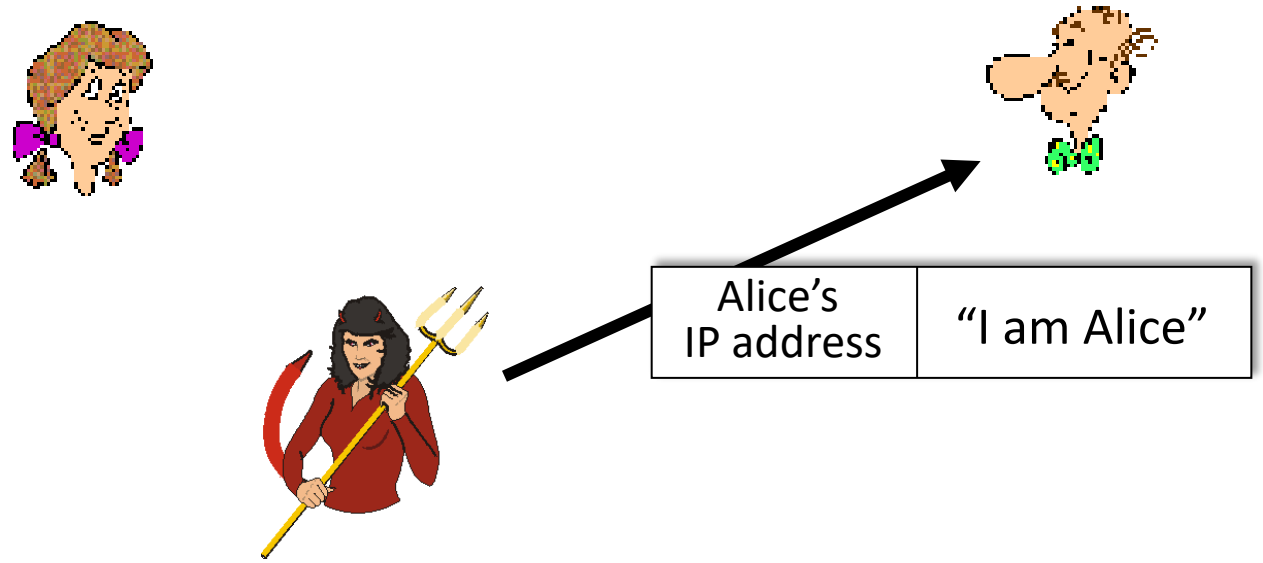



*failure scenario??*

# Authentication: another try

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

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

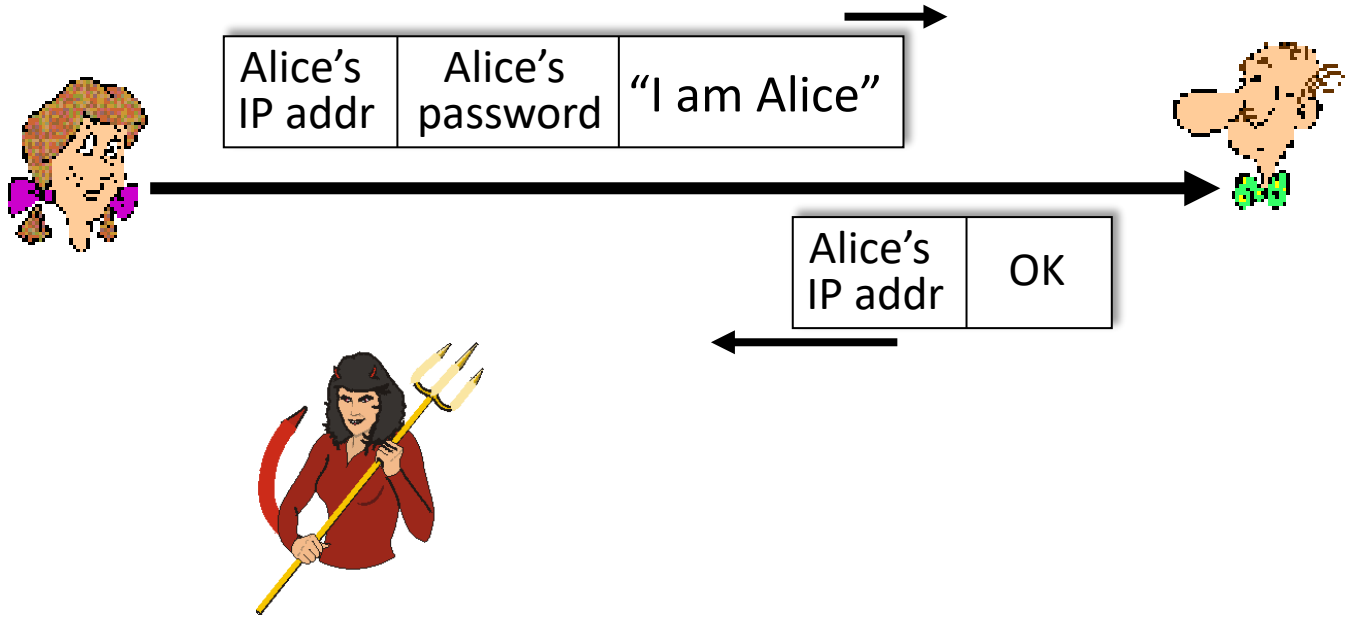



*Trudy can create a packet "spoofing" Alice's address*

# Authentication: a third try

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

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

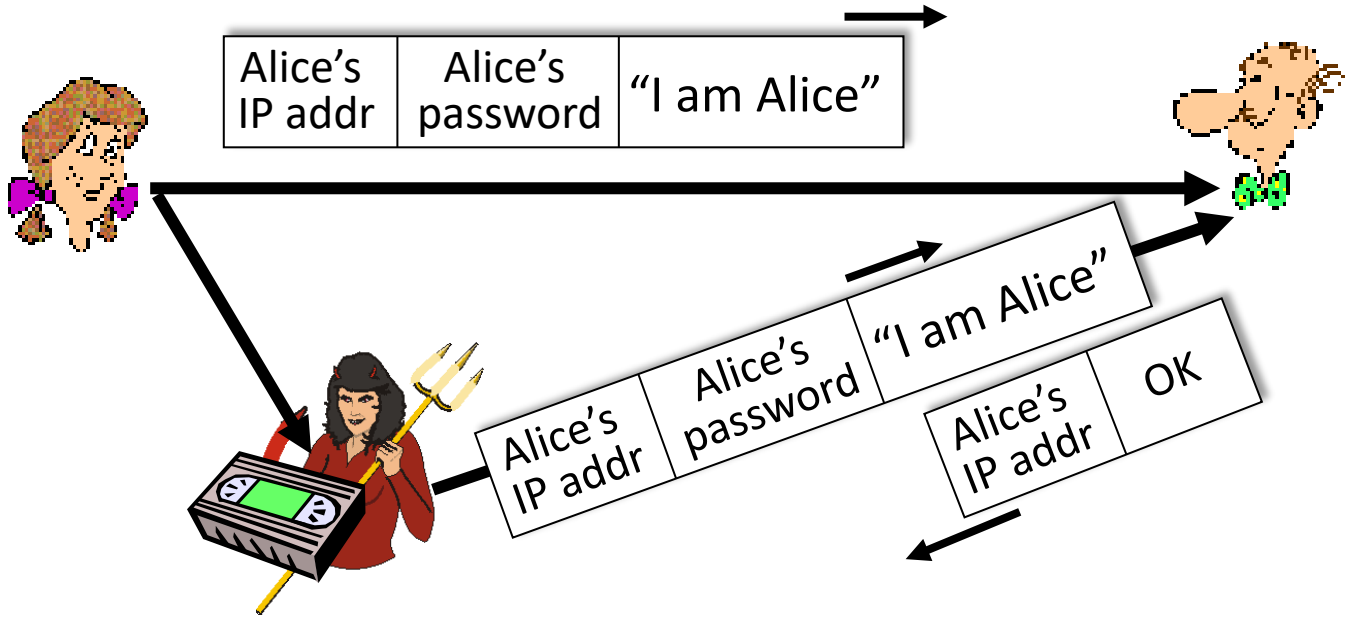



*failure scenario??*

# Authentication: a third try

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

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

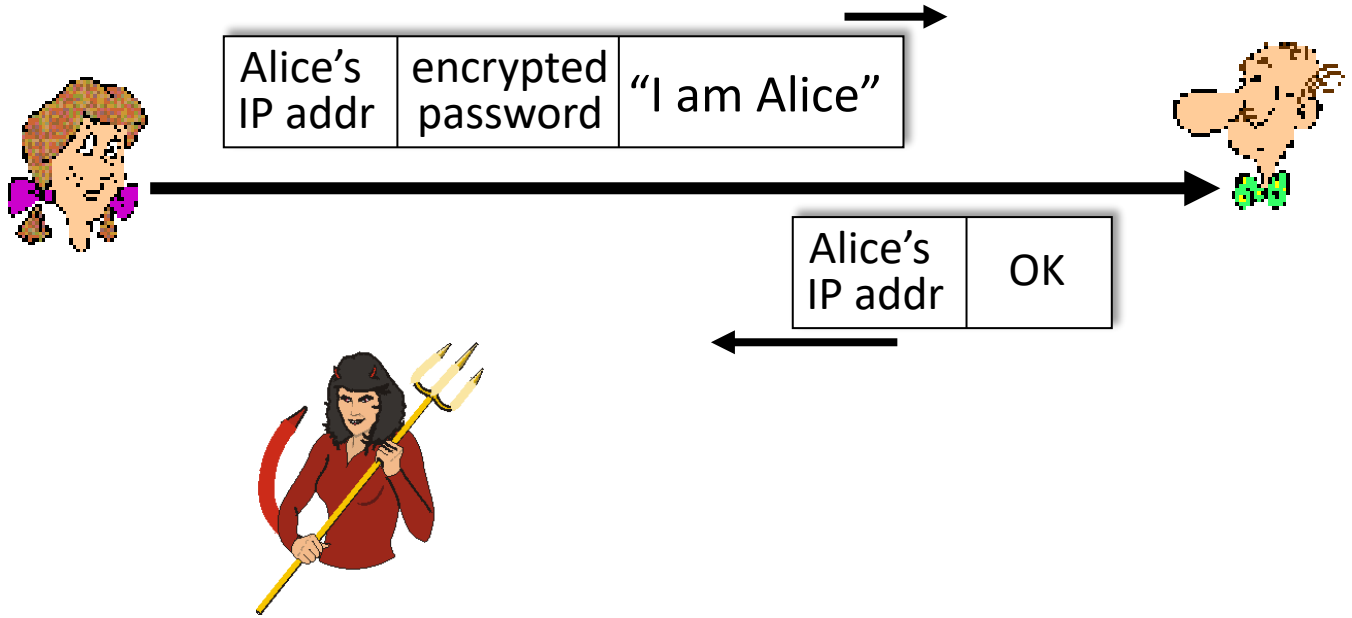



*playback attack:*
*Trudy records Alice's packet and later plays it back to Bob*

# Authentication: a modified third try

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

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

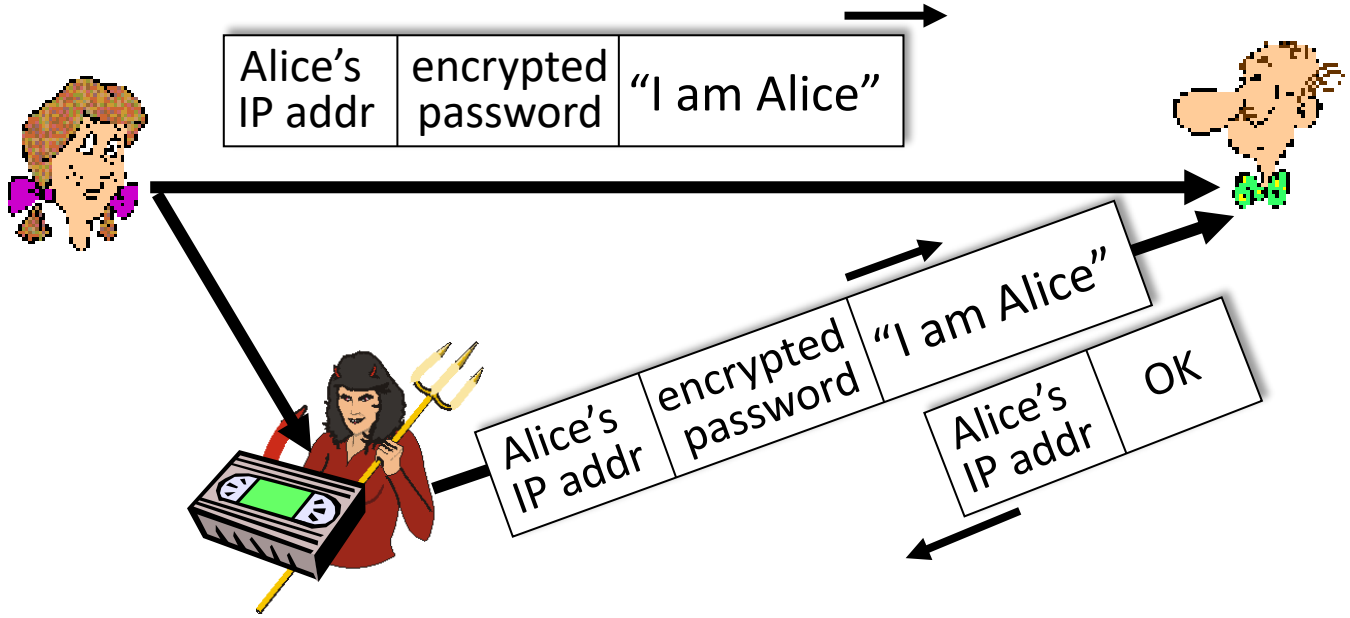



*failure scenario??*

# Authentication: a modified third try

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

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




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

# Authentication: a fourth try

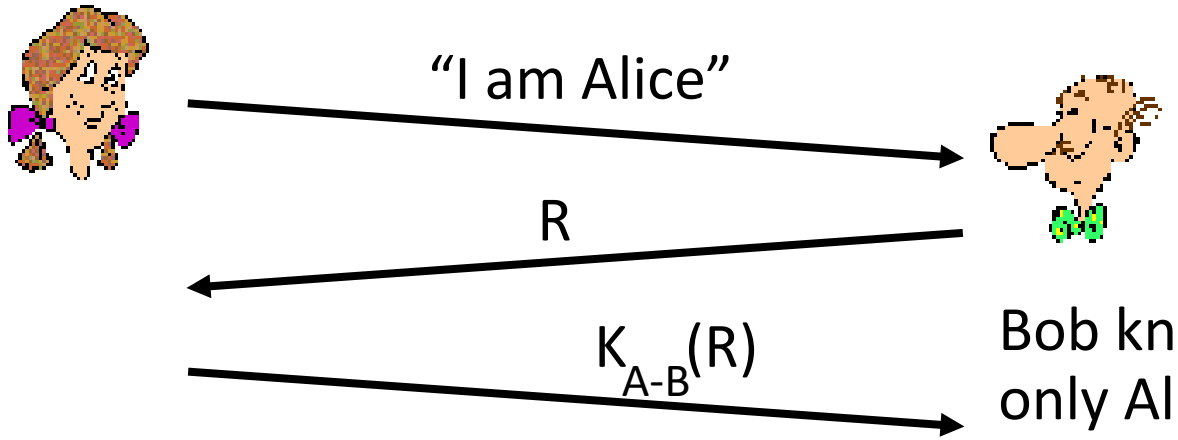
Goal: avoid playback attack

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

To address the playback attack, it expires quickly to ensure the message is fresh

protocol ap4.0: to prove Alice "live", Bob sends Alice nonce, R

- Alice must return R, encrypted with shared secret key




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

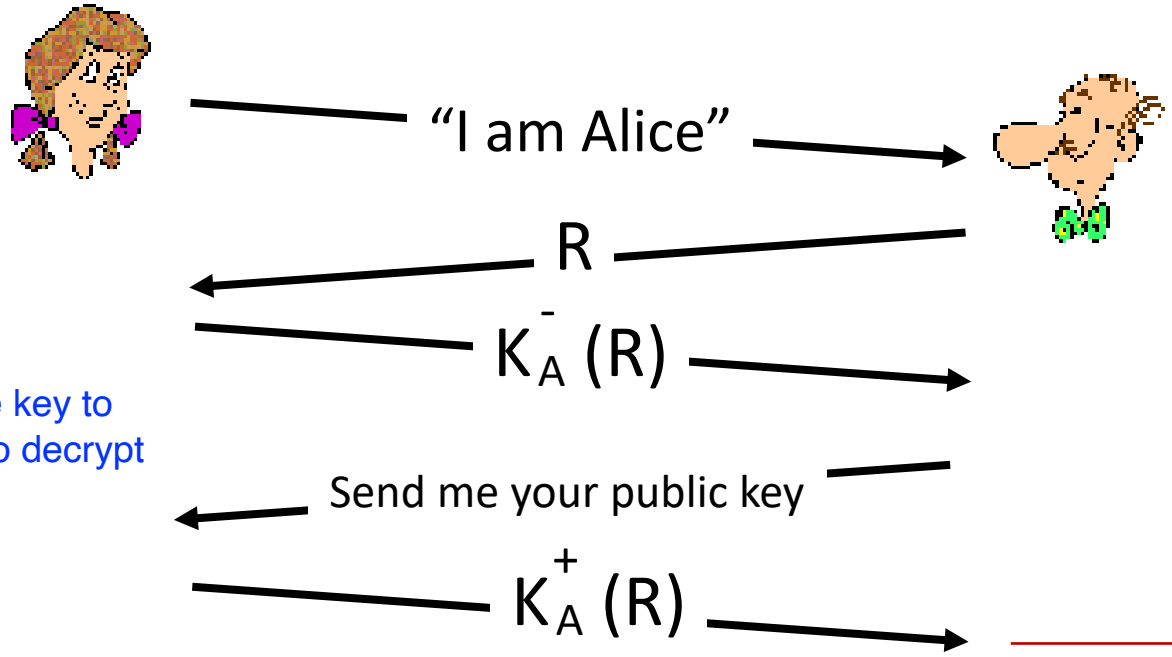
*Failures, drawbacks?*

Symmetric keys must be securely exchanged beforehand, which can be a logistical challenge, especially when parties have not previously met or communicated securely.

# Authentication: ap5.0

ap4.0 requires shared symmetric key - can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography




Flipping the public key encryption, using private key to encrypt and public key to decrypt

Bob computes

$K_A^+(K_A^-(R)) = R$

and knows only Alice could have the private key, that encrypted R such that

$K_A^+(K_A^-(R)) = R$

# Authentication: ap5.0 – there's still a flaw!

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

I am Alice

*Where are mistakes made here?*

I am Alice

R

$K_T^-(R)$

Send me your public key

$K_T^+$

Bob computes $K_T^+(K_T^-(R)) = R$, authenticating Trudy as Alice

R

$K_A^-(R)$

Send me your public key

$K_A^+$

Trudy recovers m:

$m = K_T^-(K_T^+(m))$

sends m to Alice encrypted with Alice's public key

$K_T^+(m)$

Bob sends a personal message, m to Alice

Trudy recovers Bob's m:

$m = K_A^-(K_A^+(m))$

$K_A^+(m)$

and she and Bob meet a week later in person and discuss m, not knowing Trudy knows m

# Chapter 2 outline

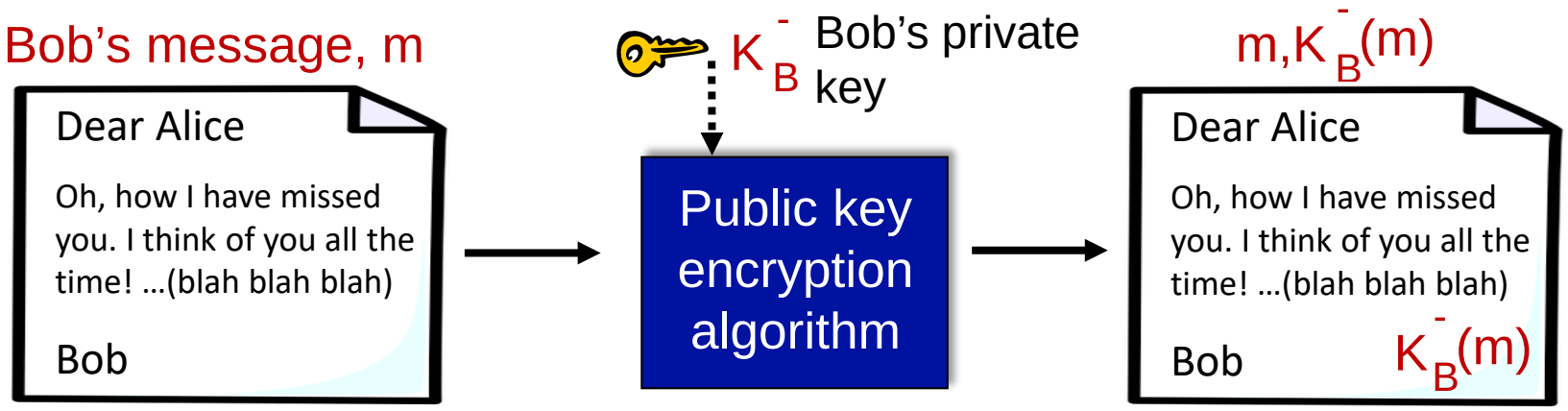
- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

# Digital signatures

Authentication, message integrity

## cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document: 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
- 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

- suppose Alice receives msg m, with signature: m, $K_B^-(m)$
- 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 the decyphered text mateches the original message, then it's true.
- 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:

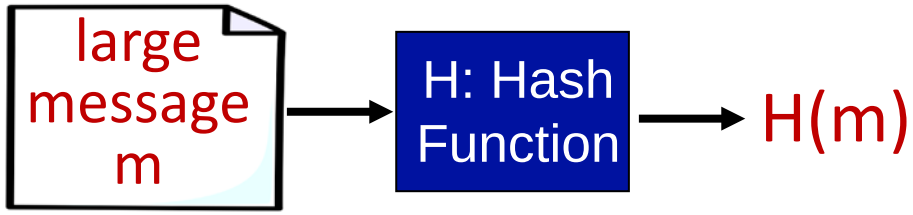
- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

# 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)*

# 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

Sender calculate the check sum before sending the message, leaving the sum in the header of the message.

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

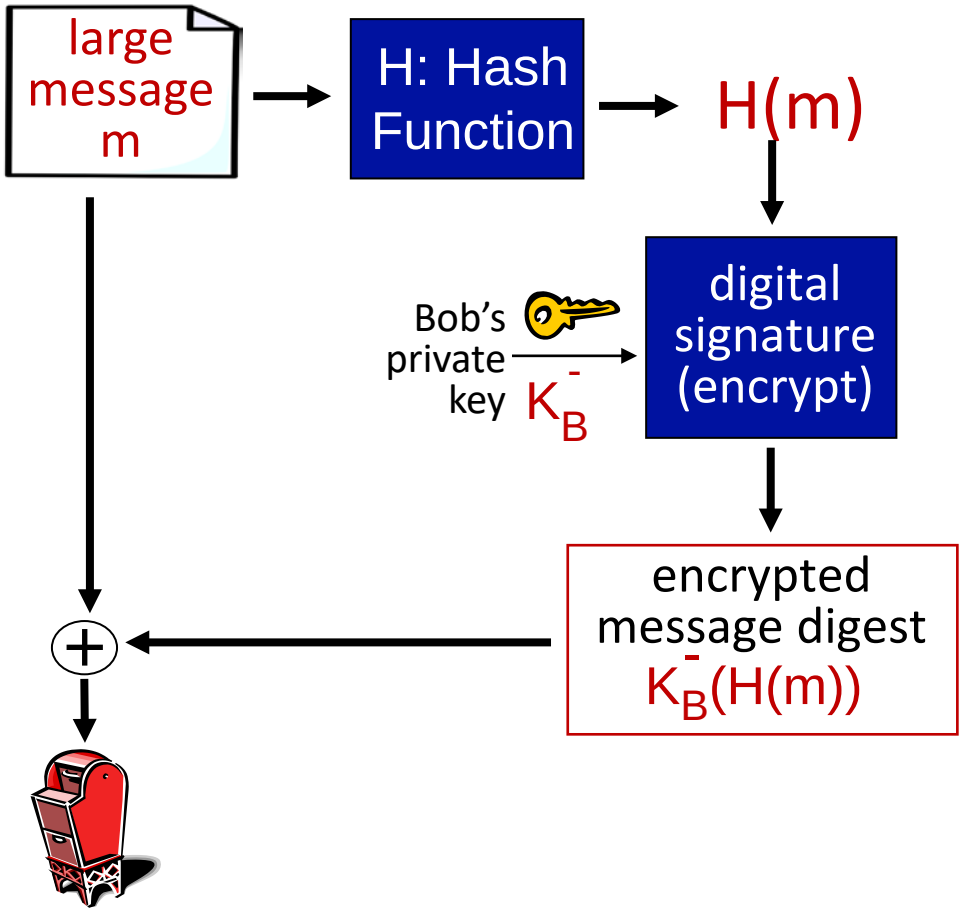
| message | ASCII format |
|---|---|
| I O U 1 | 49 4F 55 31 |
| 0 0 . 9 | 30 30 2E 39 |
| 9 B O B | 39 42 D2 42 |
| | B2 C1 D2 AC |

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

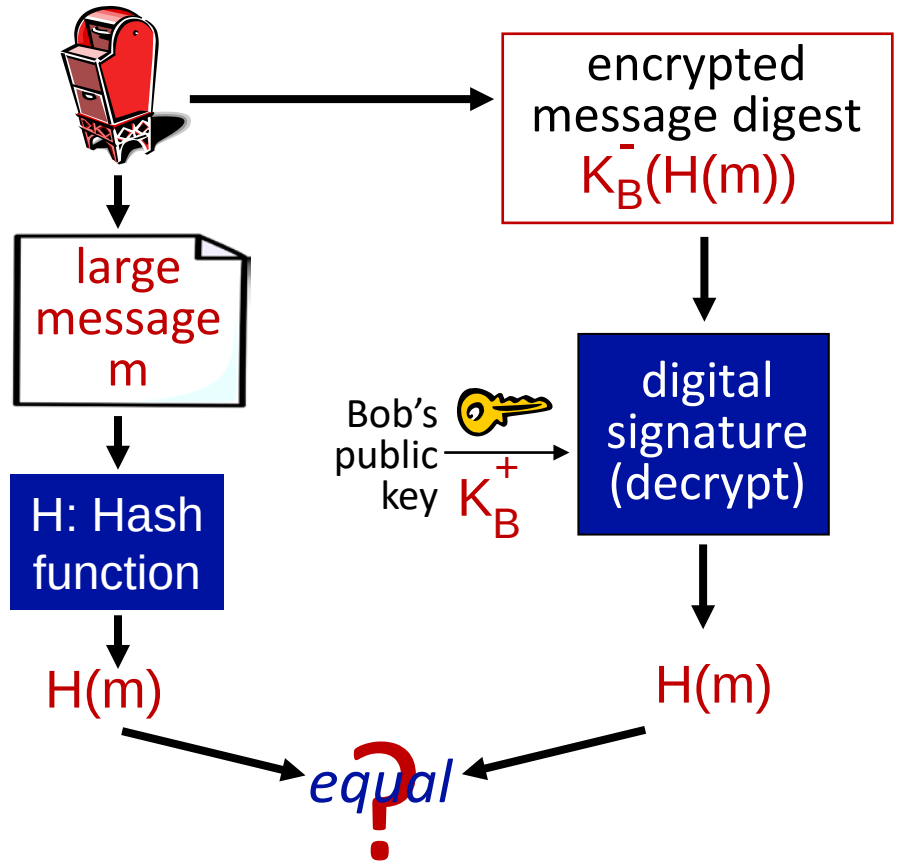
*different messages but identical checksums!*

# Digital signature = signed message digest

Bob sends digitally signed message:




Alice verifies signature, integrity of digitally signed message:

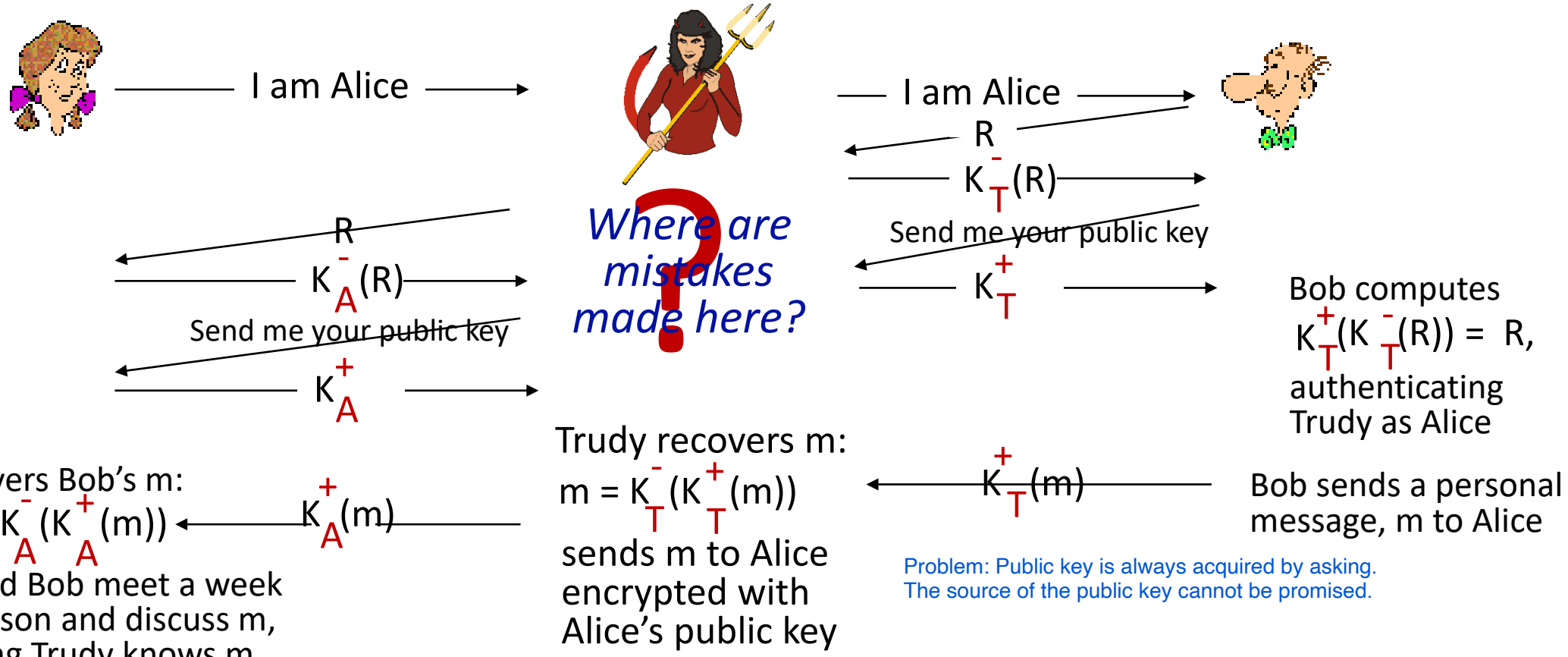


# Hash function algorithms

- MD5 hash function widely used (RFC 1321) Request for Comment
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x
- SHA-1 is also used Secure Hash Algorithm
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

# Authentication: ap5.0 – let's fix it!!

Recall the problem: Trudy poses as Alice (to Bob) and as Bob (to Alice)

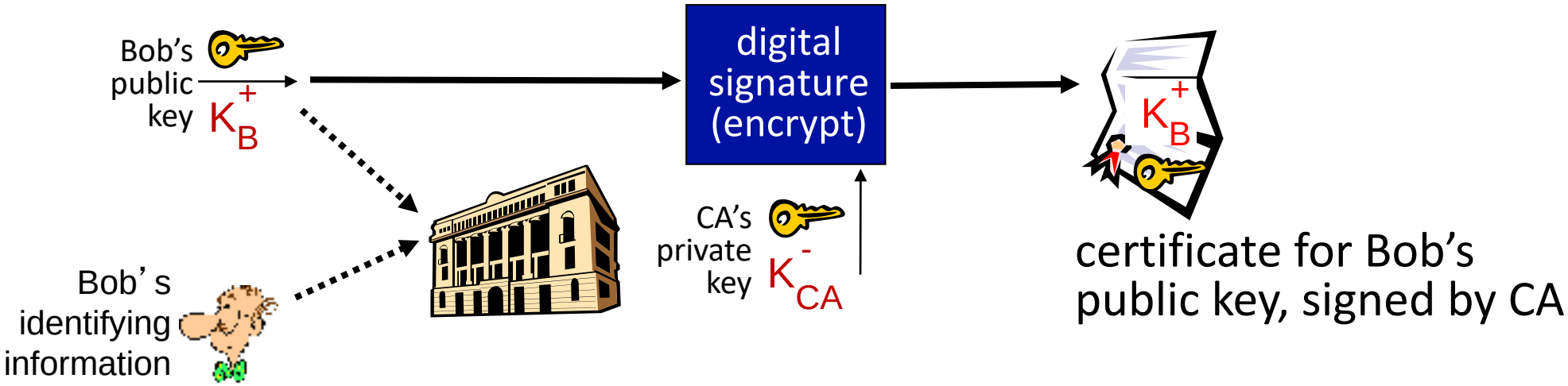


# Need for certified public keys

- motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    *Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob*
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni

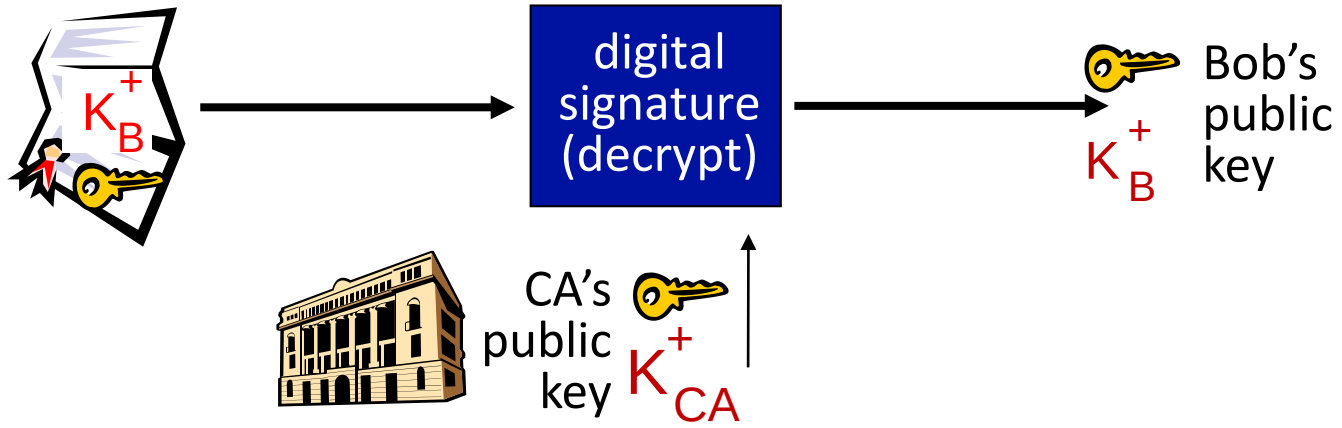
# Public key Certification Authorities (CA)

- certification authority (CA): binds public key to particular entity, E
- entity (person, website, router) registers its public key with CE provides "proof of identity" to CA
  - CA creates certificate binding identity E to E's public key
  - certificate containing E's public key digitally signed by CA: CA says "this is E's public key"



# Public key Certification Authorities (CA)

- 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

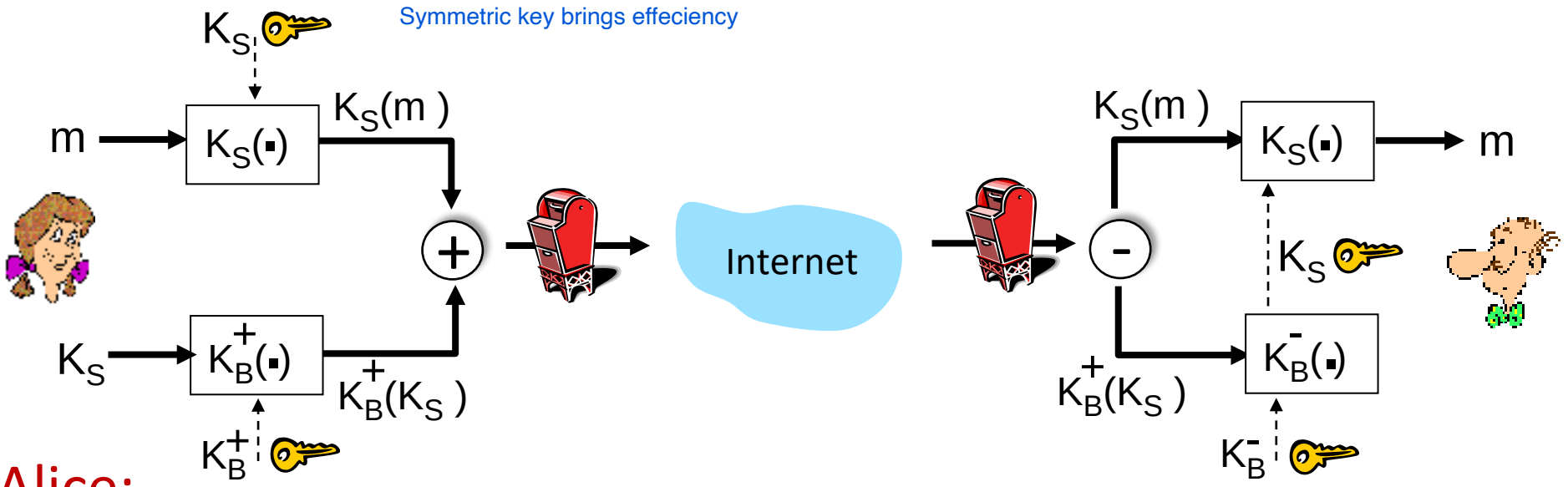


# Chapter 2 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- **Securing e-mail**
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

# Secure e-mail: confidentiality

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

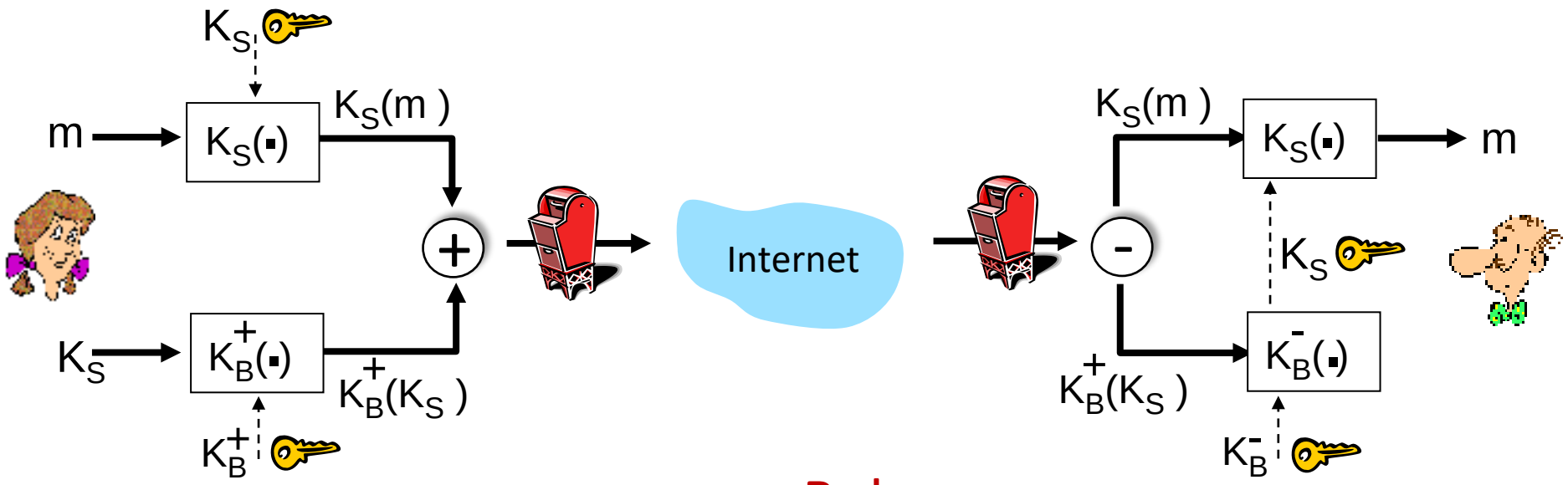



Alice:

- 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: confidentiality (more)

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




Use public/private keys to increase the confidentiality of the Ks

## Bob:

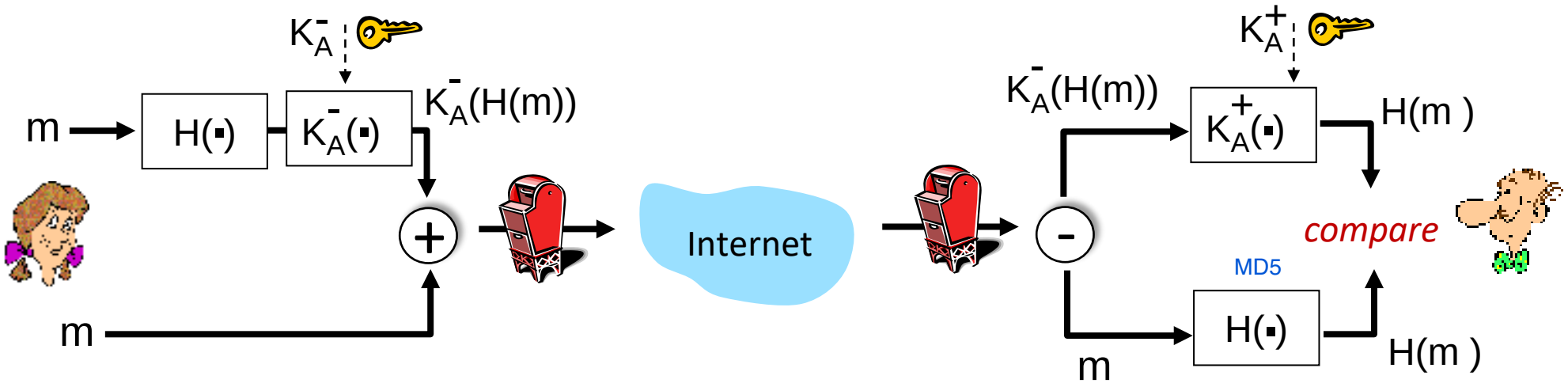
- uses his private key to decrypt and recover $K_S$
- uses $K_S$ to decrypt $K_S(m)$ to recover m

假設不考慮Confidentiality，只考慮integrity, authentication

# Secure e-mail: integrity, authentication

Alice wants to send m to Bob, with *message integrity*, *authentication*

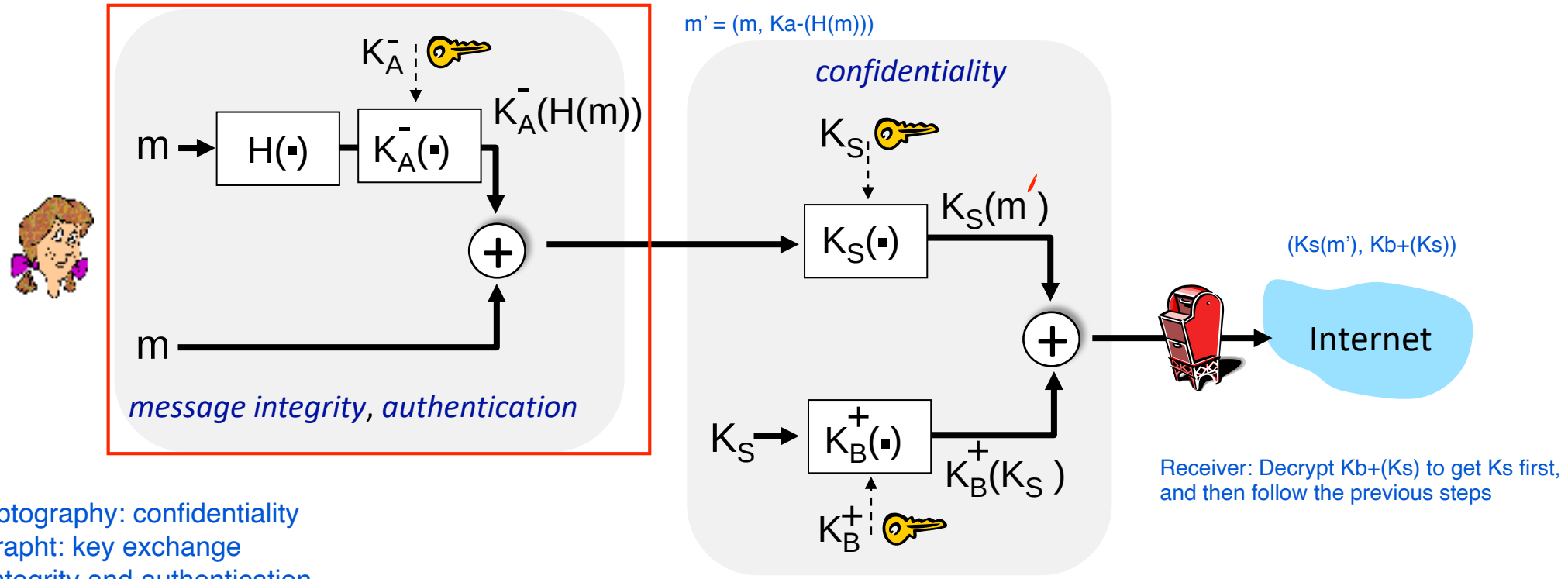
Use hash function to reduce the length of message




- Alice digitally signs hash of her message with her private key, providing integrity and authentication
- sends both message (in the clear) and digital signature

# Secure e-mail: integrity, authentication

Alice sends m to Bob, with *confidentiality, message integrity*, *authentication*




Receiver: Decrypt Kb+(Ks) to get Ks first, and then follow the previous steps

Symmetric key cryptography: confidentiality
Public key cryptographt: key exchange
Digital signature: integrity and authentication

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

*What are Bob's complementary actions?*

# Chapter 2 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- **Securing TCP connections: TLS**
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

# Transport-layer security (TLS)

TLS -> S in http"s"

- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

*all techniques we have studied!*

- history:
  - early research, implementation: secure network programming, secure sockets
  - secure socket layer (SSL) deprecated [2015]
  - TLS 1.3: RFC 8846 [2018]

# Transport-layer security (TLS)

- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

*all techniques we have studied!*

- history:
  - early research, implementation: secure network programming, secure sockets
  - secure socket layer (SSL) deprecated [2015]
  - TLS 1.3: RFC 8846 [2018]

# Transport-layer security: what's needed?

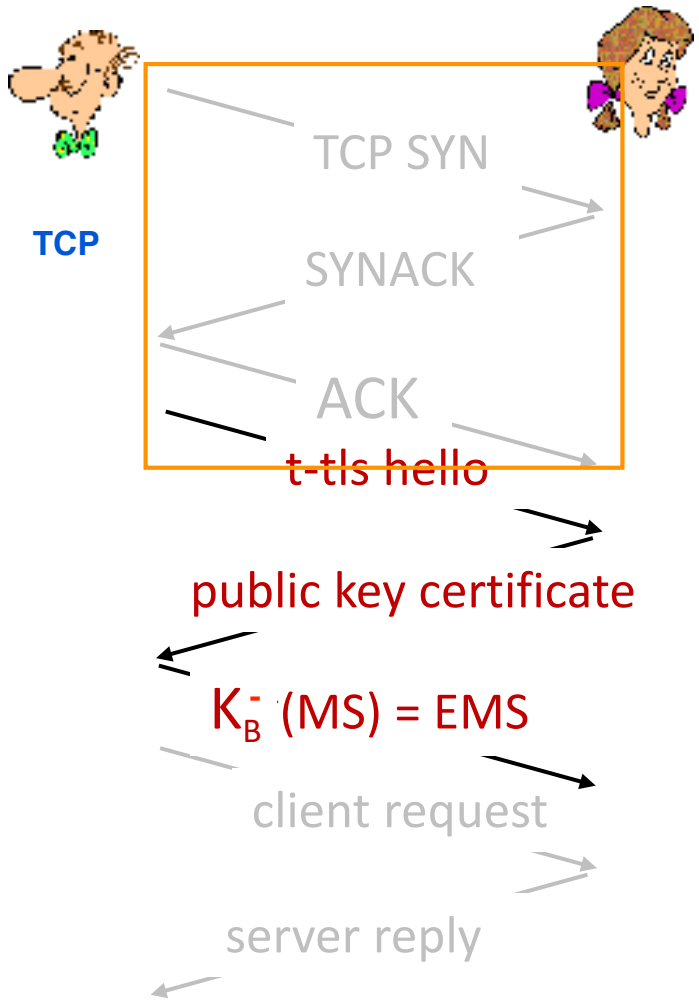
- let's *build* a toy TLS protocol, *t-tls,* to see what's needed!
- we've seen the "pieces" already:
  - handshake: Alice, Bob use their certificates, private keys to authenticate each other, exchange or create shared secret
  - key derivation: Alice, Bob use shared secret to derive set of keys
  - data transfer: stream data transfer: data as a series of records
    - not just one-time transactions

Data segmentation

  - connection closure: special messages to securely close connection

Release TCP connection resource

# t-tls: initial handshake




## t-tls handshake phase:

- Bob establishes TCP connection with Alice
- Bob verifies that Alice is really Alice
  Bob sends hello message.
  Alice replies with public key certificate.
- Bob sends Alice a master secret key (MS), used to generate all other keys for TLS session
- potential issues:
  Round Trip Time
  - 3 RTT before client can start receiving data (including TCP handshake)

# t-tls: cryptographic keys

Generate different keys with Master Secret Key

- considered bad to use same key for more than one cryptographic function
  - different keys for message authentication code (MAC) and encryption

Ensure message integrity

hash (m + s) = MAC

- four keys:
  - $K_c$ : encryption key for data sent from client to server — For confidentiality
  - $M_c$ : MAC key for data sent from client to server — Message integrity
  - $K_s$ : encryption key for data sent from server to client — For confidentiality
  - $M_s$ : MAC key for data sent from server to client — Message integrity

sender -> client
receiver -> server

- keys derived from key derivation function (KDF)
  - takes master secret and (possibly) some additional random data to create new keys

# t-tls: encrypting data

- recall: TCP provides data *byte stream* abstraction
- Q: can we encrypt data in-stream as written into TCP socket?
  - *A:* where would MAC go? If at end, no message integrity until all data received and connection closed!
  - *solution:* break stream in series of "records"
    - each client-to-server record carries a MAC, created using $M_c$
    - receiver can act on each record as it arrives
- t-tls record encrypted using symmetric key, $K_c$, passed to TCP:

Each record carries a Message Authentication Code (MAC), which is computed using a MAC key (Mc) shared between the client and the server.

# t-tls: encrypting data (more)

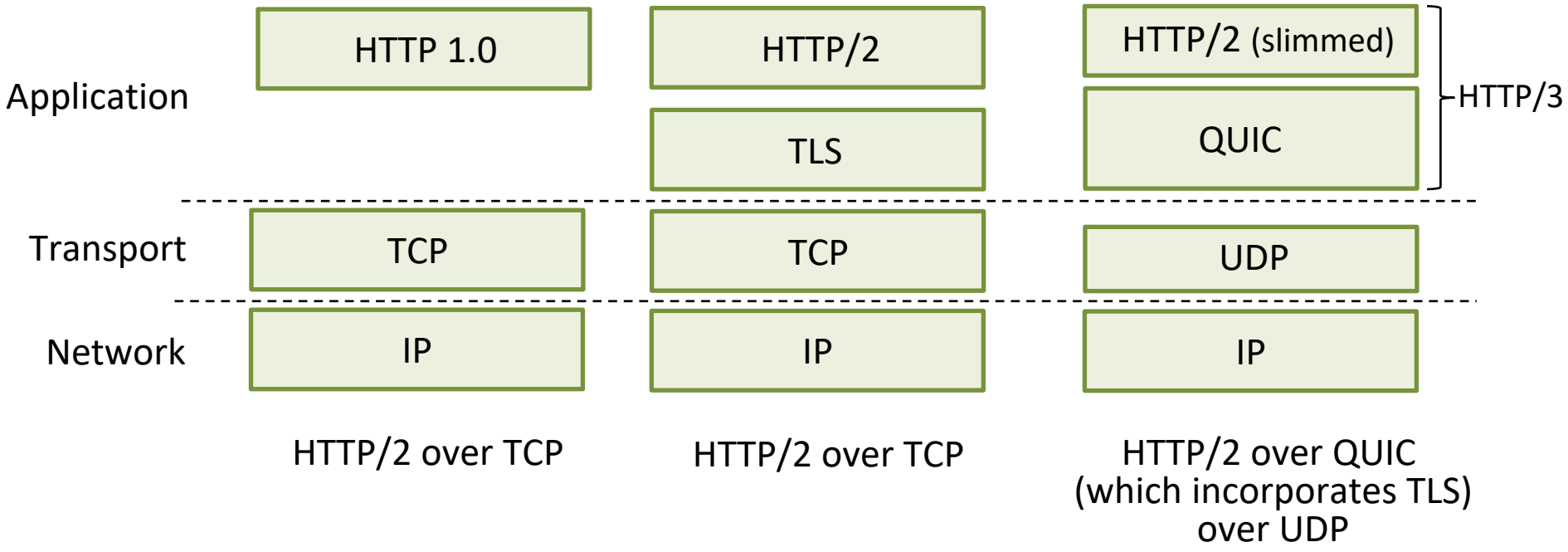
- possible attacks on data stream?
  - *re-ordering:* man-in middle intercepts TCP segments and reorders (manipulating sequence #s in unencrypted TCP header)
  - *replay*
- solutions:
  - use TLS sequence numbers (data, TLS-seq-# incorporated into MAC)
  - use nonce in MAC calculation. It's like adding a time stamp on each record to ensure its freshness.

# t-tls: connection close

- truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is
- solution: record types, with one type for closure
  - type 0 for data; type 1 for close
- MAC now computed using data, type, sequence #

# Transport-layer security (TLS)

- TLS provides an API that *any* application can use
- an HTTP view of TLS:

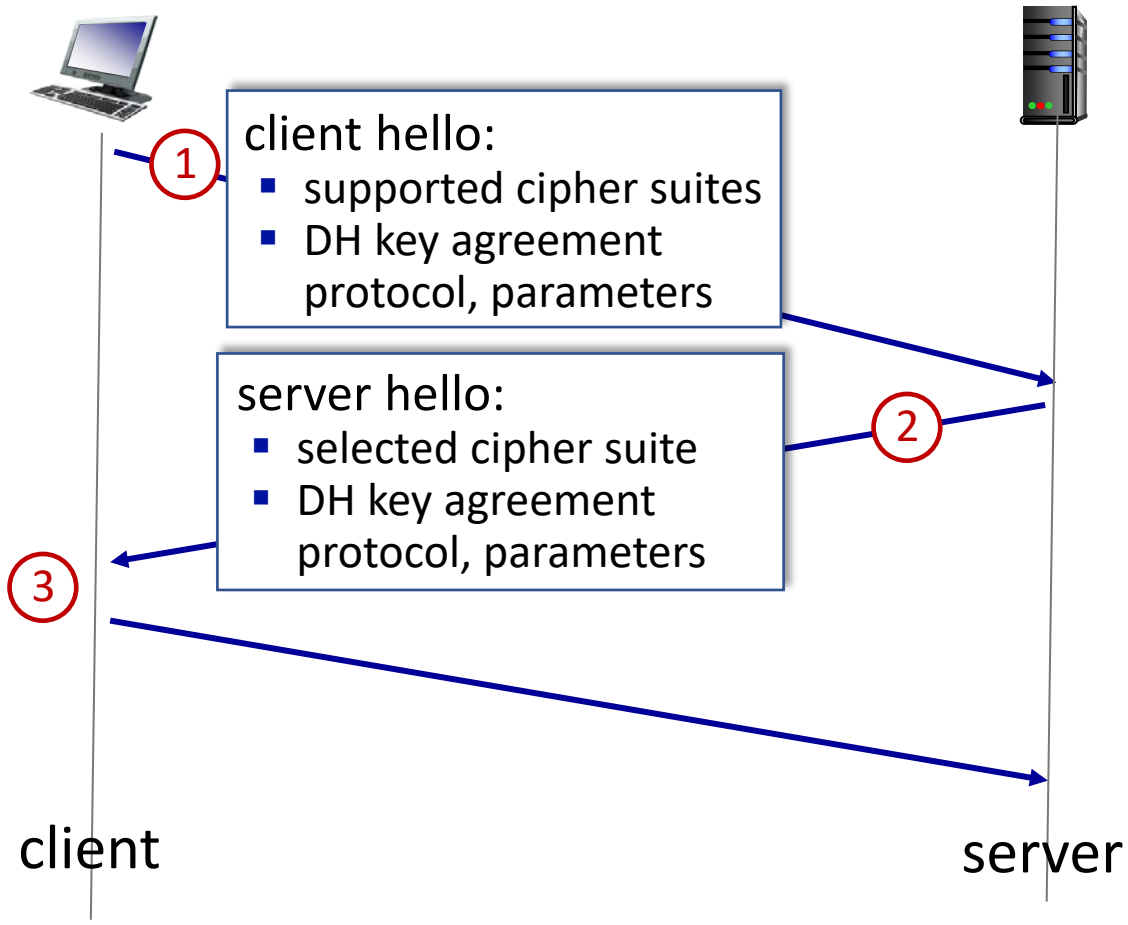


# TLS: 1.3 cipher suite

- "cipher suite": algorithms that can be used for key generation, encryption, MAC, digital signature
- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
  - only 5 choices, rather than 37 choices
  - *requires* Diffie-Hellman (DH) for key exchange, rather than DH or RSA
  - combined encryption and authentication algorithm ("authenticated encryption") for data rather than serial encryption, authentication
    - 4 based on AES
  - HMAC uses SHA (256 or 284) cryptographic hash function

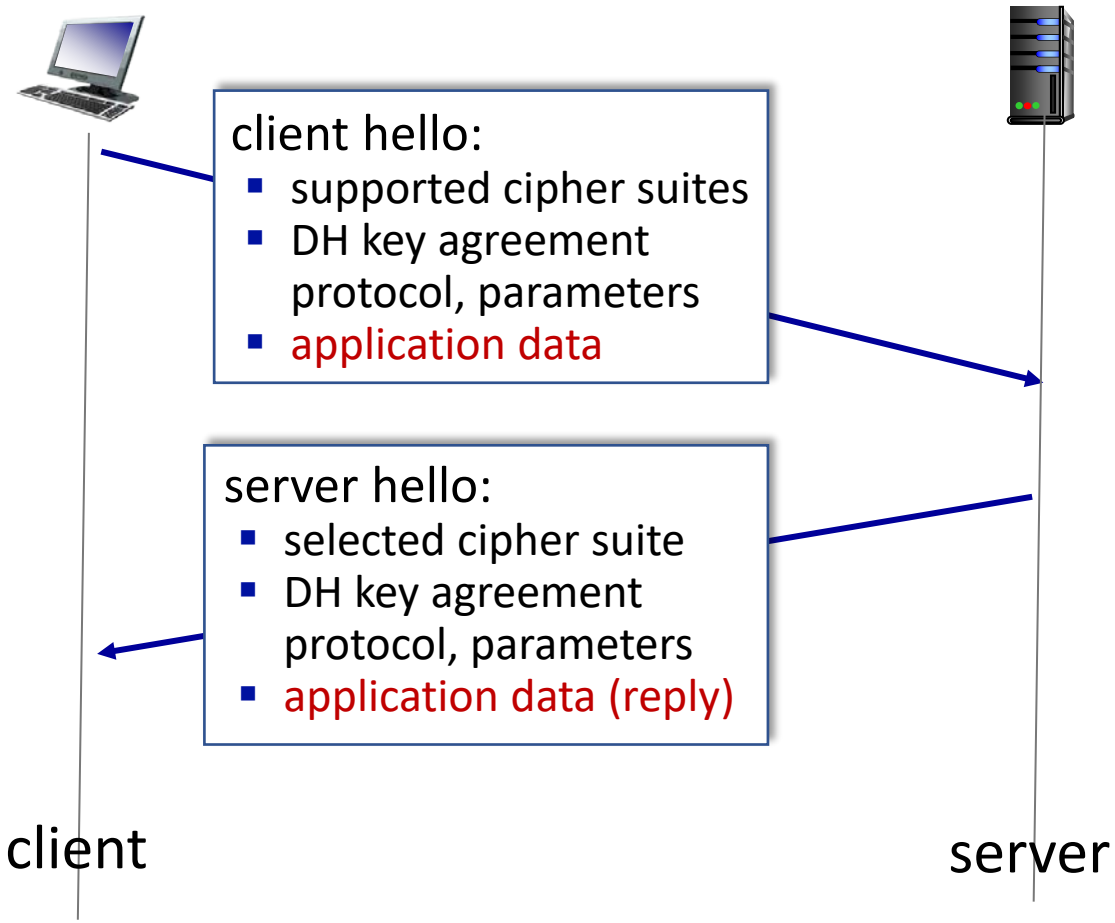
# TLS 1.3 handshake: 1 RTT

This is the standard handshake when the client is connecting to a server for the first time (i.e., no prior session exists).




1. client TLS hello msg:
   - *guesses* key agreement protocol, parameters
   - indicates cipher suites it supports
2. server TLS hello msg chooses
   - key agreement protocol, parameters
   - cipher suite
   - server-signed certificate
3. client:
   - checks server certificate
   - generates key
   - can now make application request (e.g.., HTTPS GET)

# TLS 1.3 handshake: 0 RTT




- initial hello message contains encrypted application data!
  - "resuming" earlier connection between client and server
  - application data encrypted using "resumption master secret" from earlier connection
- vulnerable to replay attacks!
  - maybe OK for get HTTP GET or client requests not modifying server state

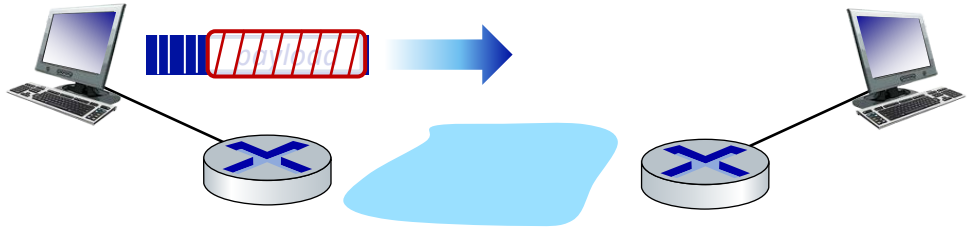
# Chapter 2 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- **Network layer security: IPsec**
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

# IP Sec

Run on network layer, support all application using IP

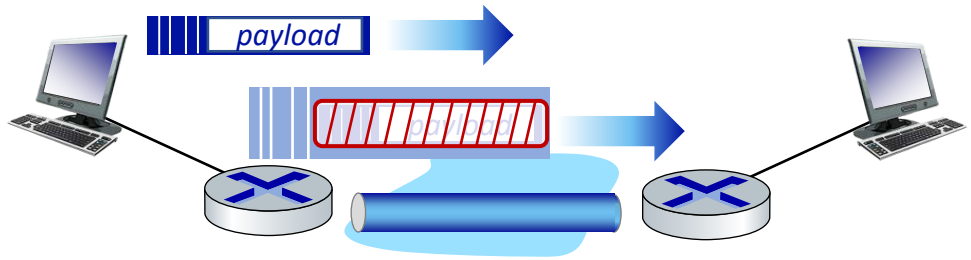
- provides datagram-level encryption, authentication, integrity
  - for both user traffic and control traffic (e.g., BGP, DNS messages)
- two "modes":




transport mode:

- *only* datagram *payload* is encrypted, authenticated

The payload is encrypted and authenticated, but the IP header which contains address info is visible from outside




**Higher security**

tunnel mode:

It's how virtual private network (VPN) works.

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

**Because the entire datagram is encrypted (including the origin IP header), a new IP header is needed to identify the destination.**

# Two IPsec protocols

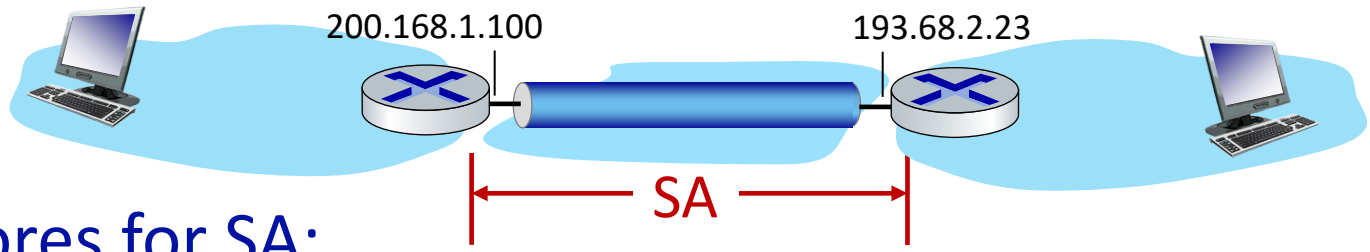
- Authentication Header (AH) protocol [RFC 4302]
  - provides source authentication & data integrity but *not* confidentiality
- Encapsulation Security Protocol (ESP) [RFC 4303]
  - provides source authentication, data integrity, *and confidentiality*
  - more widely used than AH

# Security associations (SAs)

Tunnel

- before sending data, security association (SA) established from sending to receiving entity (directional)
- ending, receiving entitles maintain *state information* about SA
  - recall: TCP endpoints also maintain state info
  - IP is connectionless; IPsec is connection-oriented!

Both are gate way rounter




SA is single direction

R1 stores for SA:

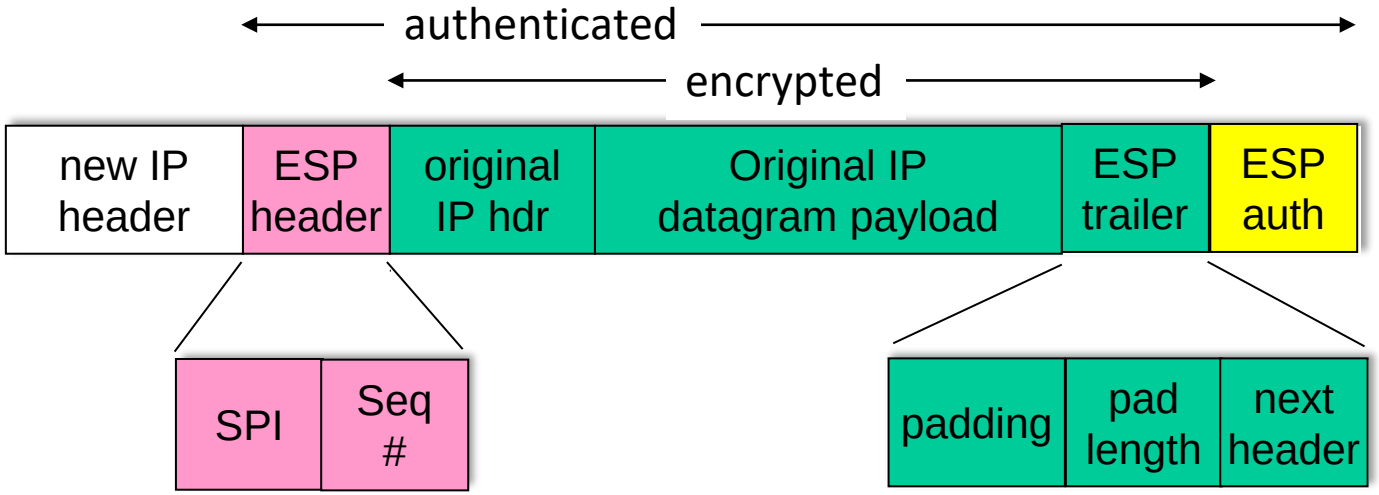
- 32-bit identifier: *Security Parameter Index (SPI)*
- origin SA interface (200.168.1.100)
- destination SA interface (193.68.2.23)
- type of encryption used
- encryption key
- type of integrity check used
- authentication key

# IPsec datagram

Headers:
Original IP Header (Inside Encrypted Payload):

Source IP: 192.168.1.1 (Host A).
Destination IP: 10.0.0.1 (Host B).
New IP Header (Added by Tunnel Mode):

Source IP: 203.0.113.1 (Gateway R1).
Destination IP: 203.0.113.2 (Gateway R2).




*tunnel mode ESP*

- 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 created with shared secret key

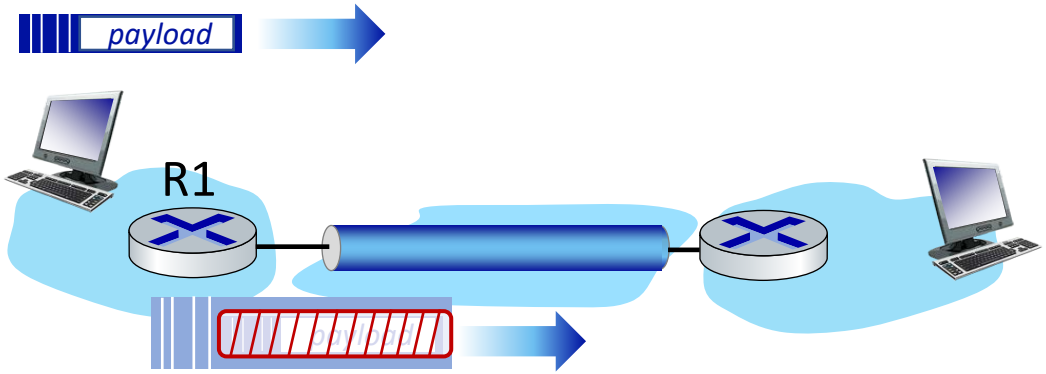
Padding: Used to maintain a fixed length for block cypher

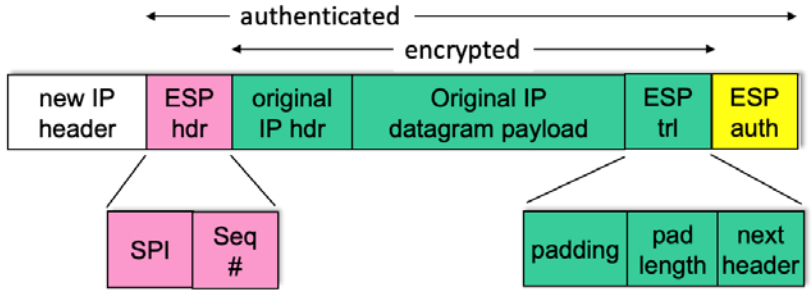
Security Parameter Index: A unique 32-bit identifier used to distinguish between multiple SAs on the same system.
Included in the header of IPsec packets to indicate which SA should be used to process the packet.

# ESP tunnel mode: actions

at R1:

- appends ESP trailer to original datagram (which includes original header fields!)
- encrypts result using algorithm & key specified by SA
- appends ESP header to front of this encrypted quantity
- creates authentication MAC using algorithm and key specified in SA
- appends MAC forming *payload*
- creates new IP header, new IP header fields, addresses to tunnel endpoint



# 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

# IPsec security databases

## Security Policy Database (SPD)

- policy: for given datagram, sender needs to know if it should use IP sec
- policy stored in security policy database (SPD)
- needs to know which SA to use
  - may use: source and destination IP address; protocol number

*SAD: "how" to do it*

## Security Assoc. Database (SAD)

- endpoint holds SA state in security association database (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, processing
- datagram accordingly.

*SPD: "what" to do*

# Summary: IPsec services

Trudy sits somewhere between R1, 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 (Pre-shared key)
  - with PKI (pubic/private keys and certificates). (Public Key Infrastructure)
- 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.

# IKE phases

- IKE has two phases
  - *phase 1:* establish bi-directional IKE SA
    - note: IKE SA different from IPsec SA
    - aka ISAKMP security association
  - *phase 2:* ISAKMP is used to securely negotiate IPsec pair of SAs
- phase 1 has two modes: aggressive mode and main mode
  - aggressive mode uses fewer messages
  - main mode provides identity protection and is more flexible

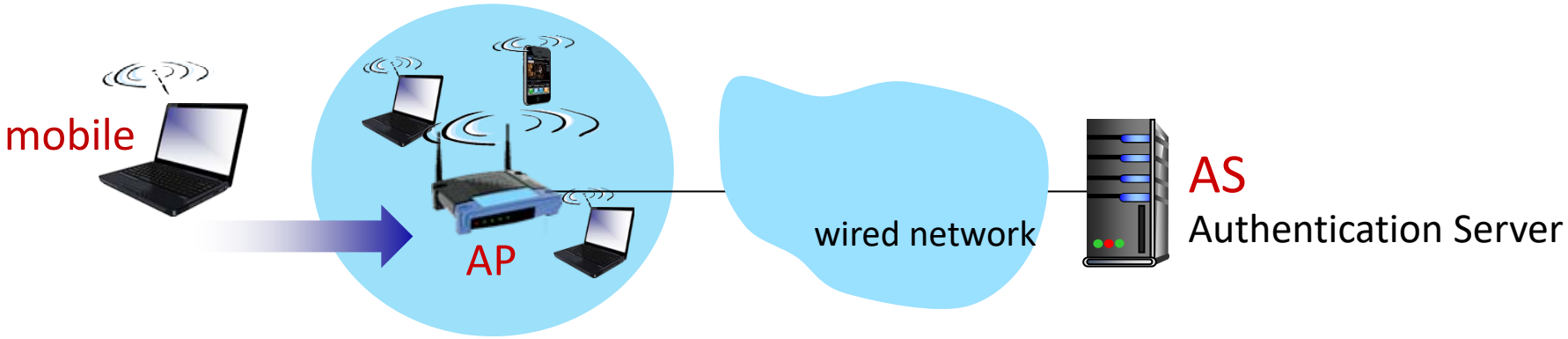
# IPsec summary

- 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

# Chapter 2 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- **Security in wireless and mobile networks**
  - 802.11 (WiFi)
  - 4G/5G
- Operational security: firewalls and IDS

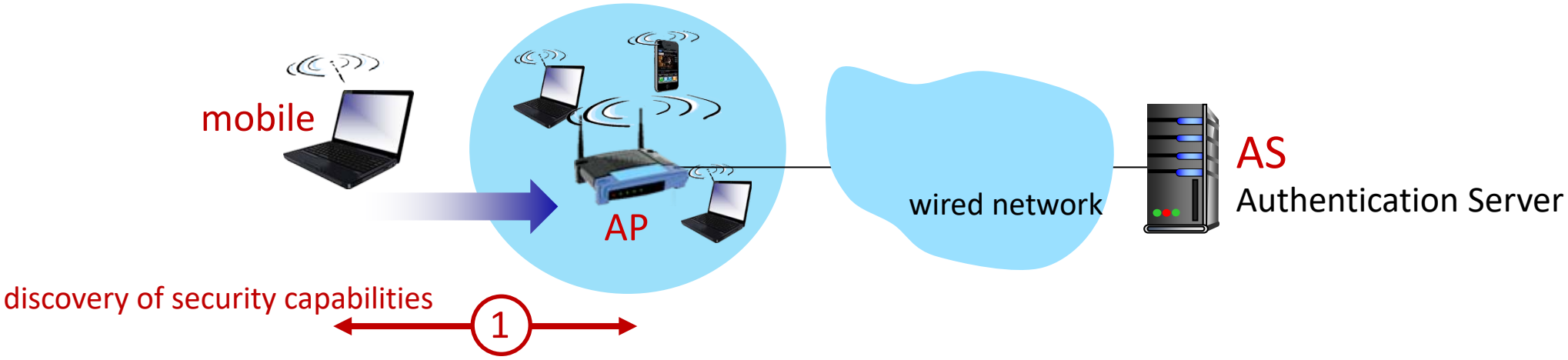
# 802.11: authentication, encryption




Arriving mobile must:

- associate with access point: (establish) communication over wireless link
- authenticate to network

# 802.11: authentication, encryption

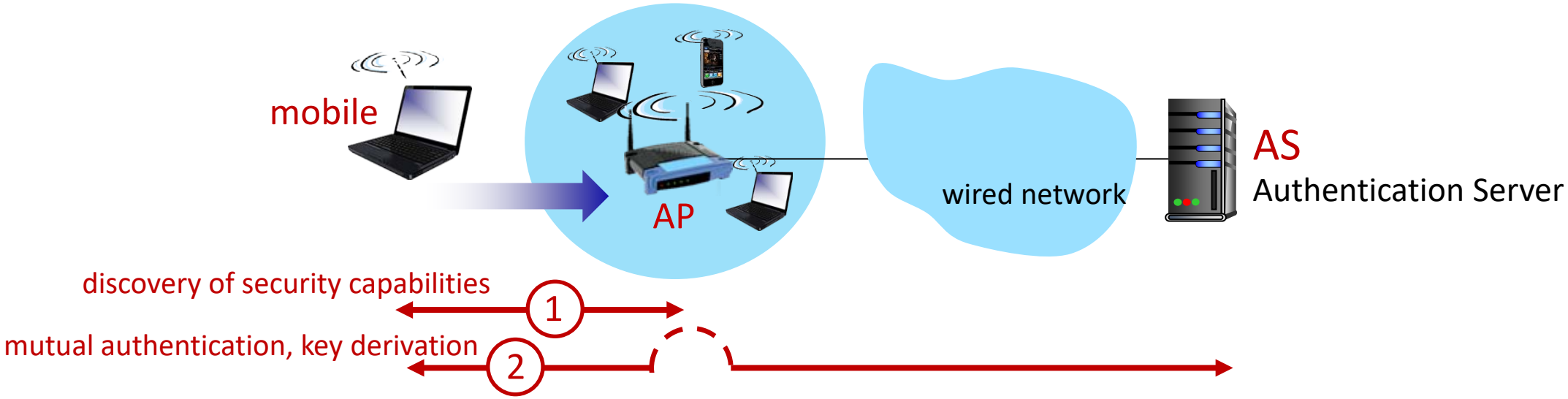



1 discovery of security capabilities:

- AP advertises its presence, forms of authentication and encryption provided
- device requests specific forms authentication, encryption desired

although device, AP already exchanging messages, device not yet authenticated, does not have encryption keys

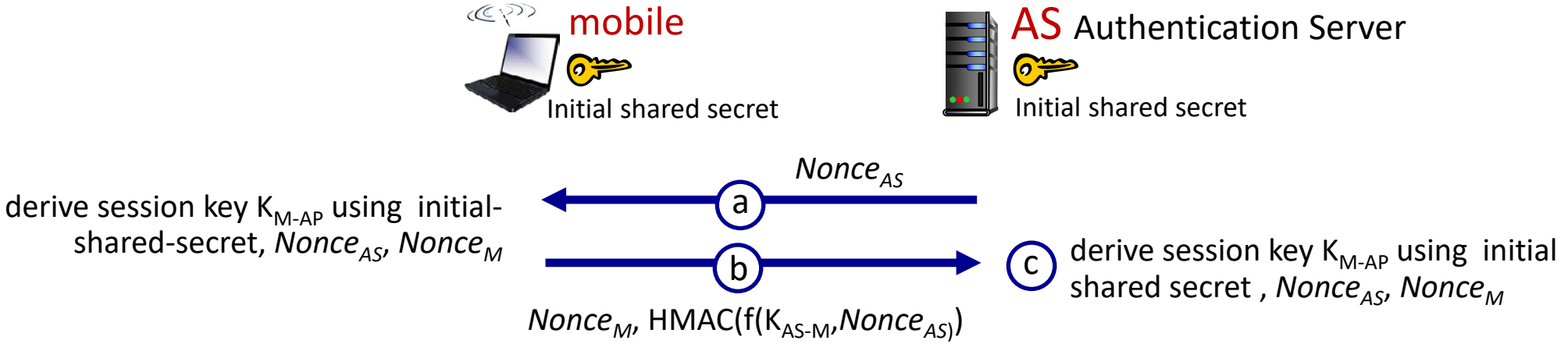
# 802.11: authentication, encryption




② mutual authentication and shared symmetric key derivation:

- AS, mobile already have shared common secret (e.g., password)
- AS, mobile use shared secret, nonces (prevent relay attacks), cryptographic hashing (ensure message integrity) to authenticating each other
- AS, mobile derive symmetric session key

# 802.11: WPA3 handshake



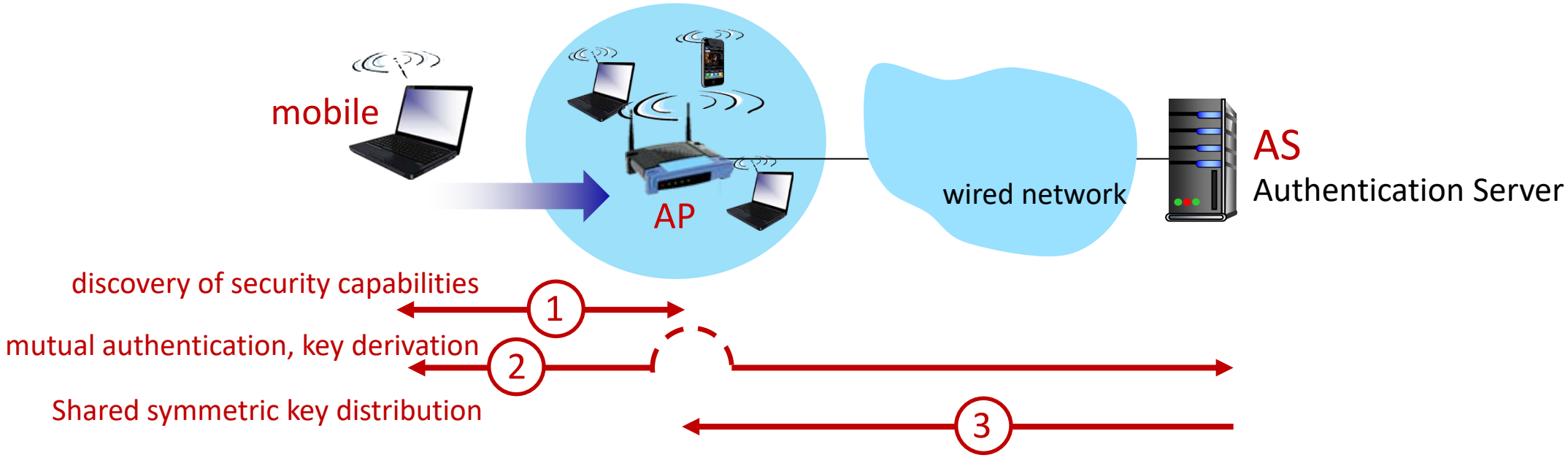

ⓐ AS generates $Nonce_{AS}$, sends to mobile

ⓑ mobile receives $Nonce_{AS}$

- generates $Nonce_M$
- generates symmetric shared session key $K_{M-AP}$ using $Nonce_{AS}$, $Nonce_M$, and initial shared secret
- sends $Nonce_M$, and HMAC-signed value using $Nonce_{AS}$ and initial shared secret

ⓒ AS derives symmetric shared session key $K_{M-AP}$

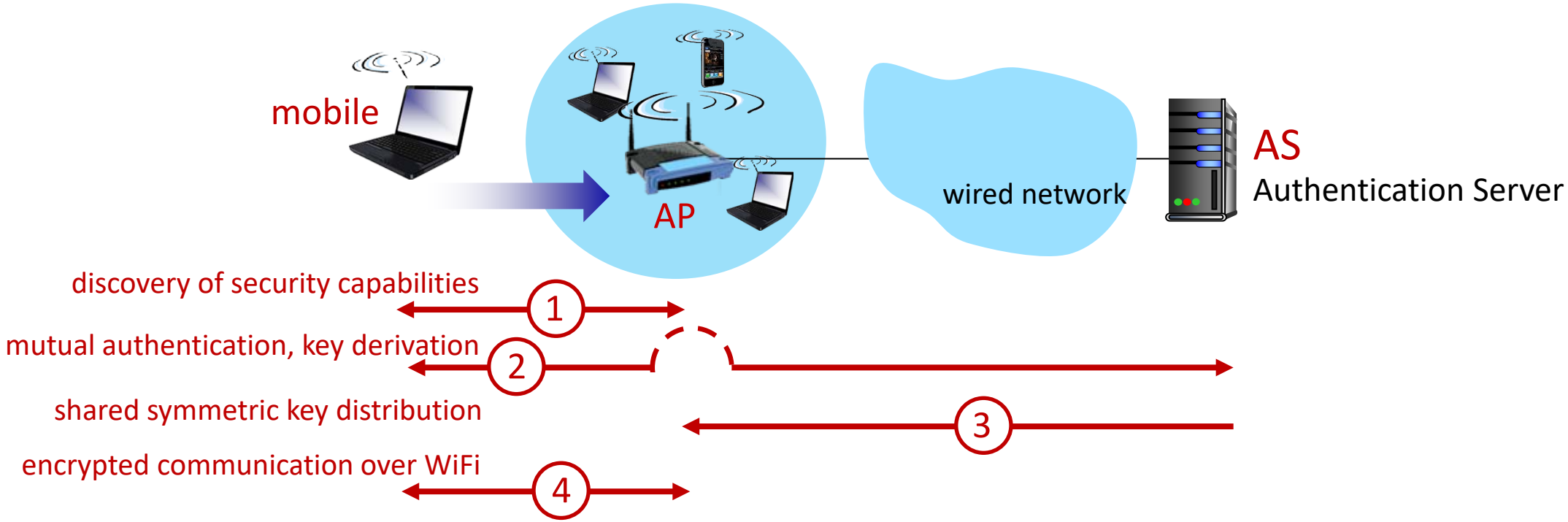
# 802.11: authentication, encryption




3 shared symmetric session key distribution (e.g., for AES encryption)

- same key derived at mobile, AS
- AS informs AP of the shared symmetric session

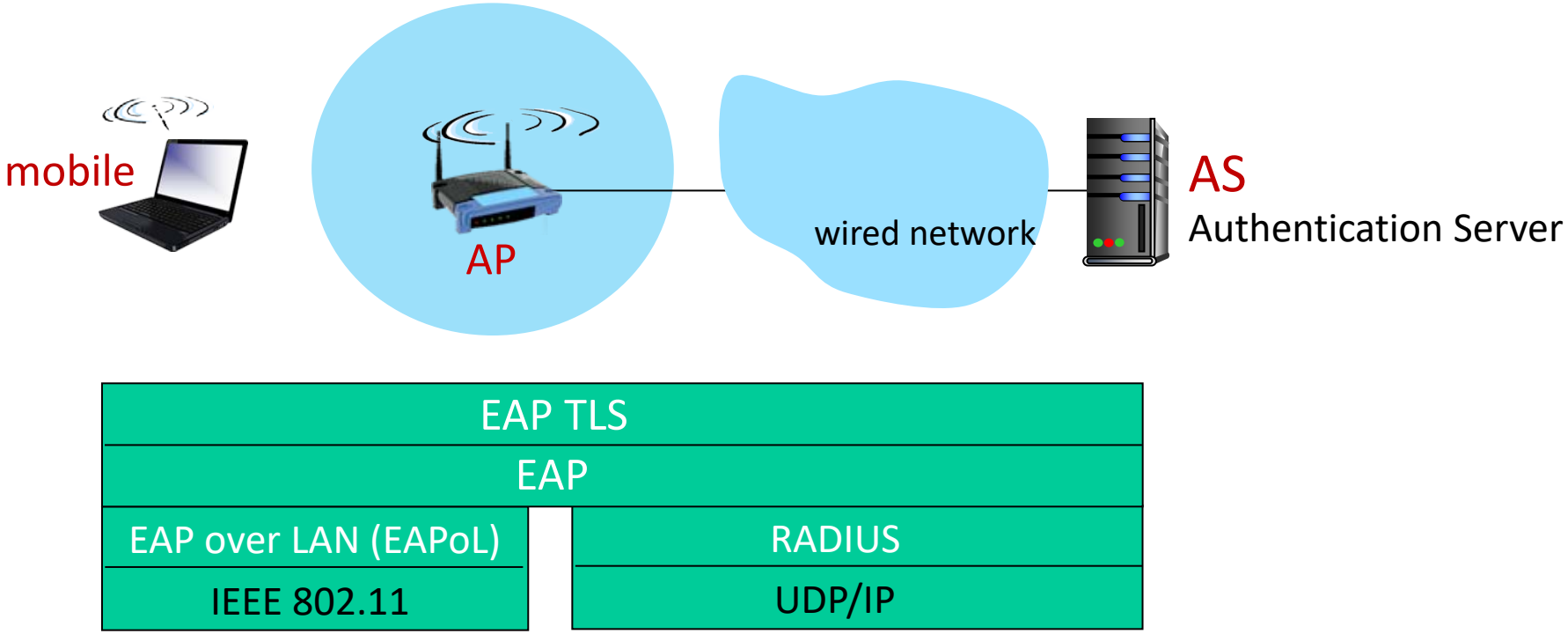
# 802.11: authentication, encryption




4 encrypted communication between mobile and remote host via AP

- same key derived at mobile, AS
- AS informs AP of the shared symmetric session

# 802.11: authentication, encryption




- Extensible Authentication Protocol (EAP) [RFC 3748] defines end-to-end request/response protocol between mobile device, AS

# Chapter 2 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- **Security in wireless and mobile networks**
  - 802.11 (WiFi)
  - 4G/5G
- Operational security: firewalls and IDS

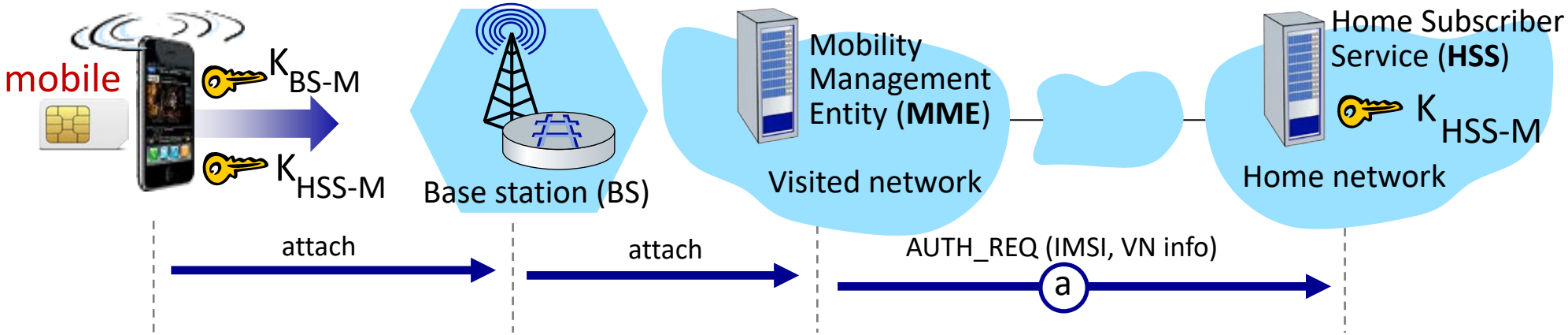
# Authentication, encryption in 4G LTE


- arriving mobile must:
  - associate with BS: (establish) communication over 4G wireless link
  - authenticate itself to network, and authenticate network
- notable differences from WiFi
  - mobile's SIMcard provides global identity, contains shared keys
  - services in visited network depend on (paid) service subscription in home network

# Authentication, encryption in 4G LTE


- mobile, BS use derived session key $K_{BS-M}$ to encrypt communications over 4G link
- MME in visited network + HHS in home network, together play role of WiFi AS
  - ultimate authenticator is HSS
  - trust and business relationship between visited and home networks

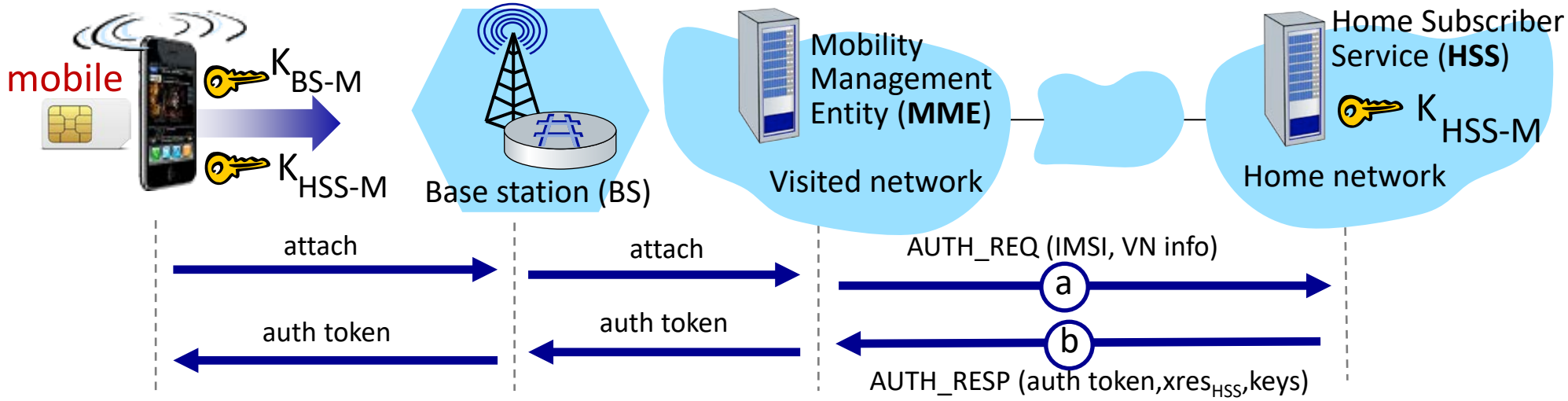
# Authentication, encryption in 4G LTE




ⓐ authentication request to home network HSS

- mobile sends attach message (containing its IMSI, visited network info) relayed from BS to visited MME to home HHS
- IMSI identifies mobile's home network

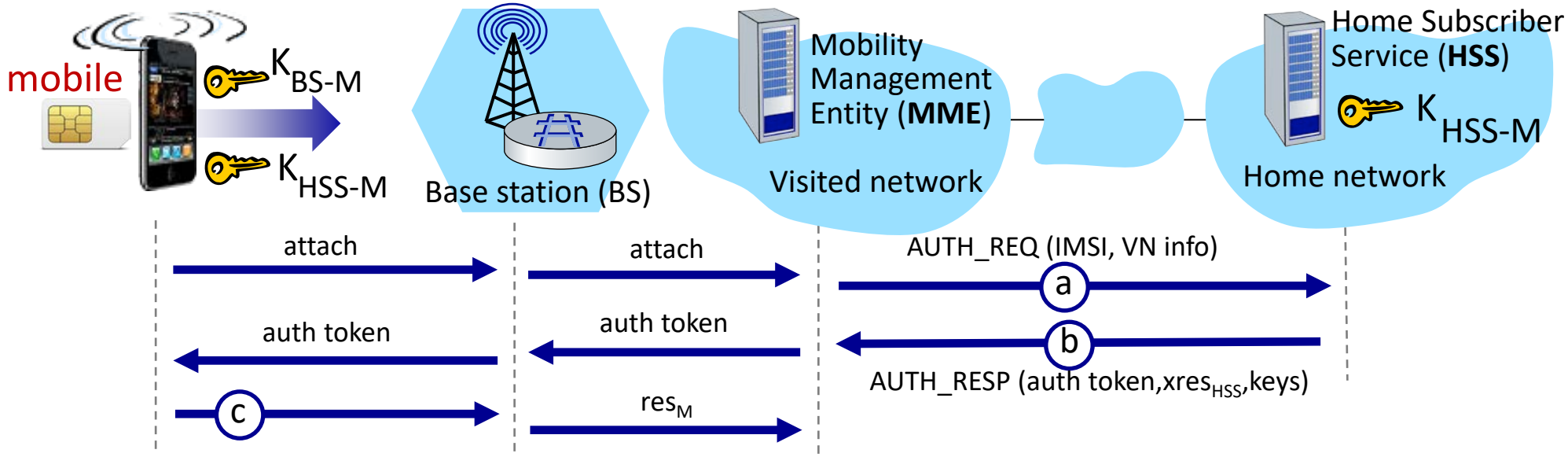
# Authentication, encryption in 4G LTE




ⓑ HSS use shared-in-advance secret key, $K_{HSS-M}$, to derive authentication token, *auth_token*, and expected authentication response token, $xres_{HSS}$

- *auth_token* contains info encrypted by HSS using $K_{HSS-M}$ , allowing mobile to know that whoever computed *auth_token* knows shared-in-advance secret
- mobile has authenticated network
- visited HSS keeps $xres_{HSS}$ for later use

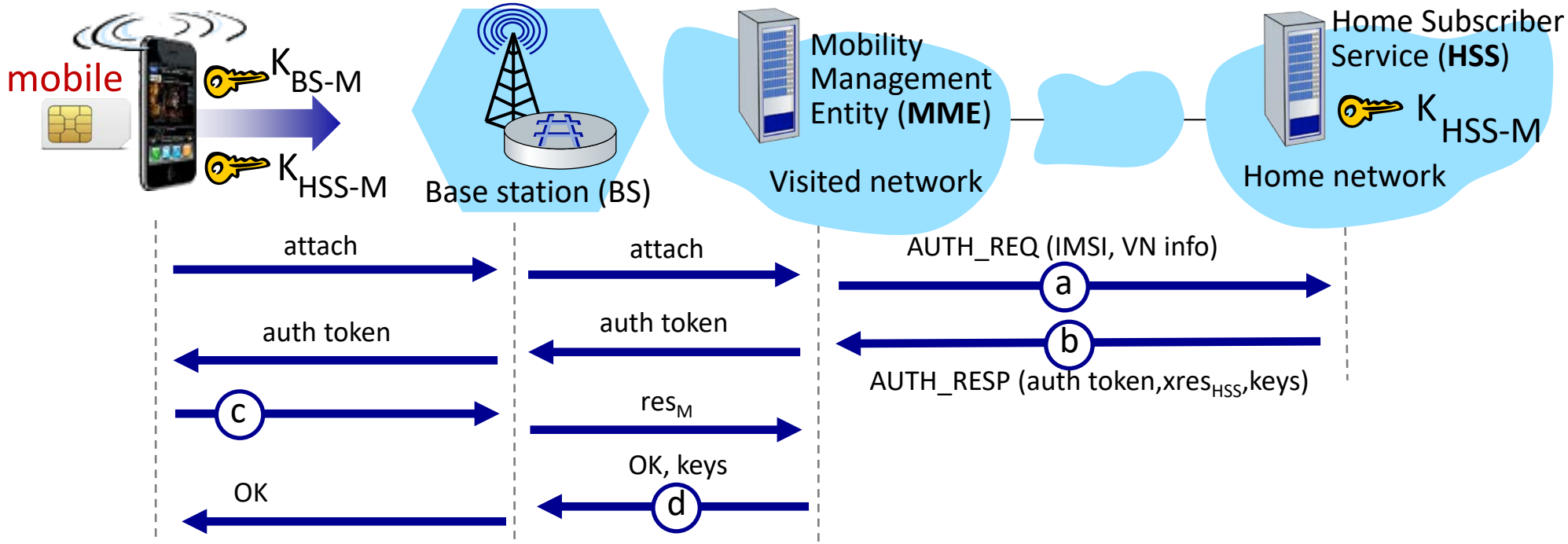
# Authentication, encryption in 4G LTE




ⓒ authentication response from mobile:

- mobile computes $res_M$ using its secret key to make same cryptographic calculation that HSS made to compute $xres_{HSS}$ and sends $res_M$ to MME

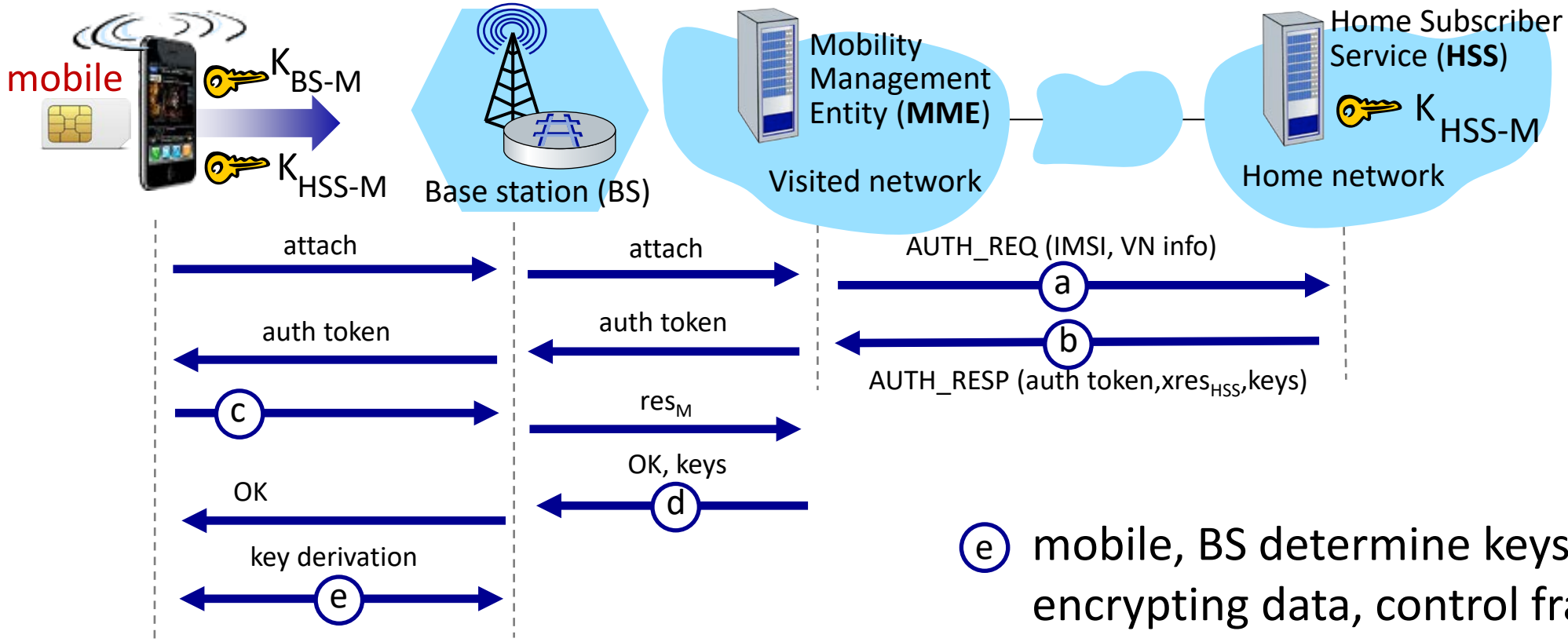
# Authentication, encryption in 4G LTE




ⓓ mobile is authenticated by network:

- MMS compares mobile-computed value of $res_M$ with the HSS-computed value of $xres_{HSS}$ . If they match, mobile is authenticated ! (why?)
- MMS informs BS that mobile is authenticated, generates keys for BS

# Authentication, encryption in 4G LTE




(e) mobile, BS determine keys for encrypting data, control frames over 4G wireless channel

- AES can be used

# Authentication, encryption: from 4G to 5G

- 4G: MME in visited network makes authentication decision
- 5G: home network provides authentication decision
  - visited MME plays “middleman” role but can still reject

- 4G: uses shared-in-advance keys
- 5G: keys not shared in advance for IoT

- 4G: device IMSI transmitted in cleartext to BS
- 5G: public key crypto used to encrypt IMSI

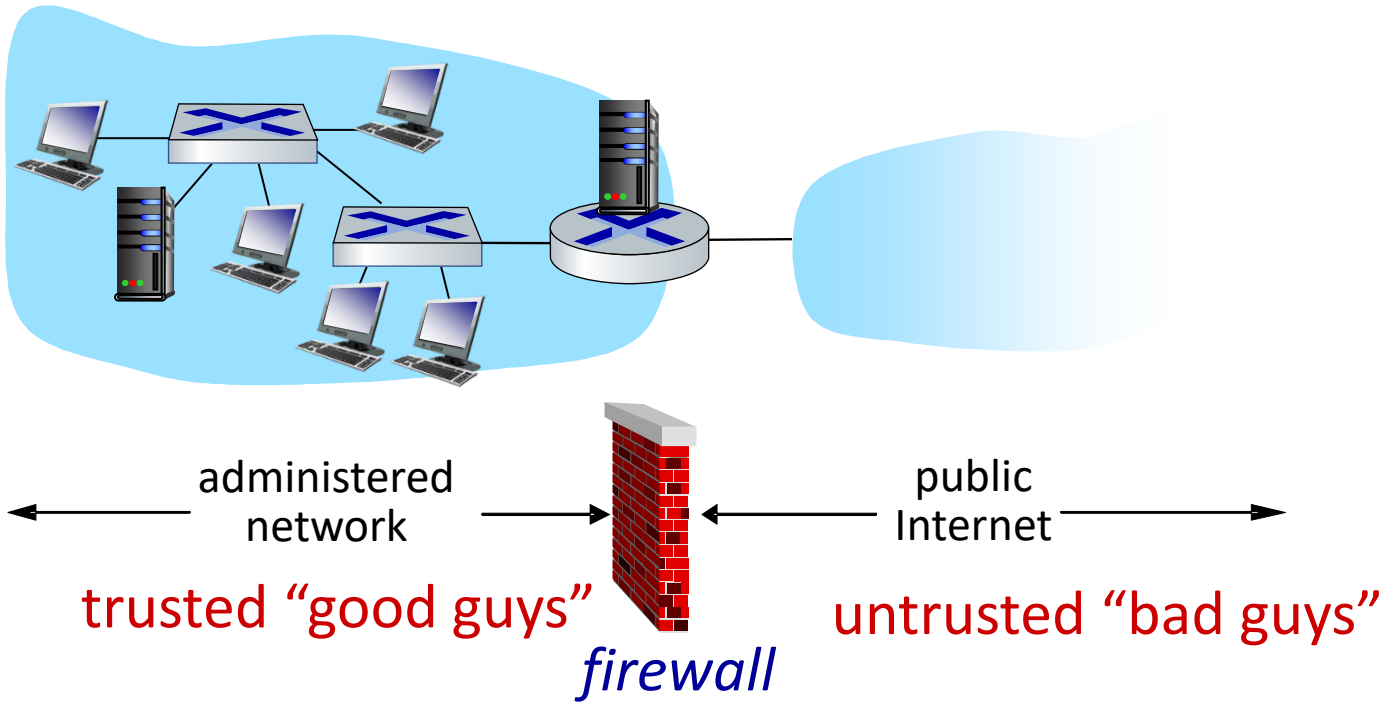
# Chapter 2 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

# Firewalls

**firewall**

isolates organization's internal network 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

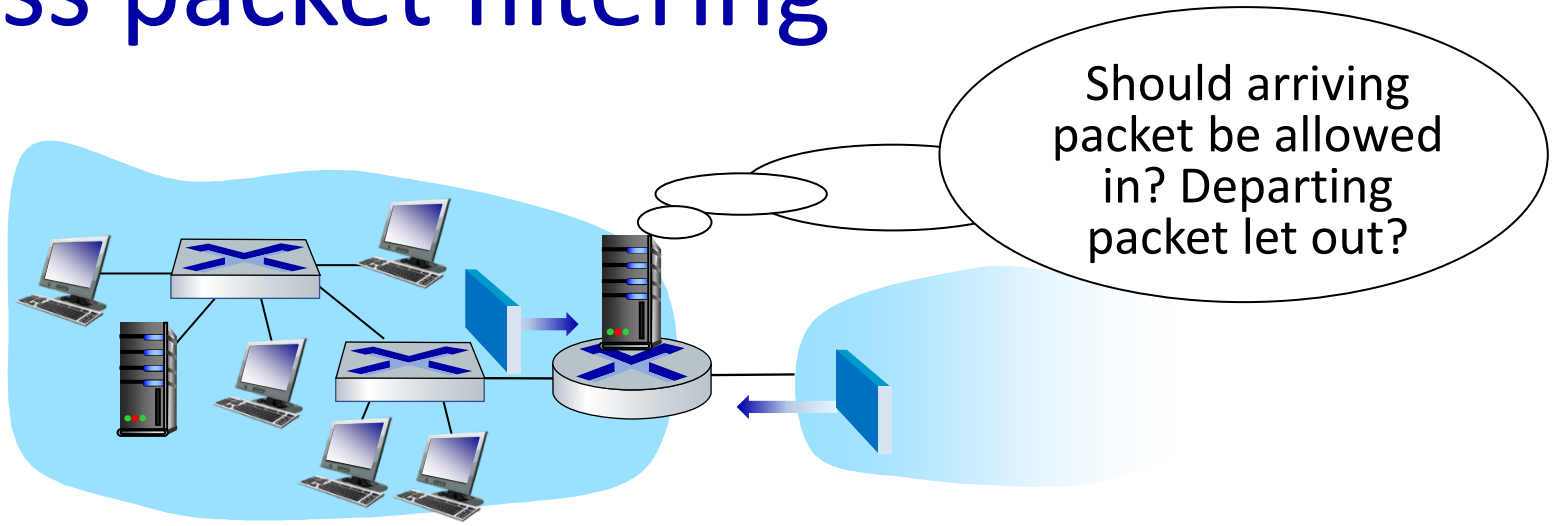
allow only authorized access to inside network

- set of authenticated users/hosts

three types of firewalls:

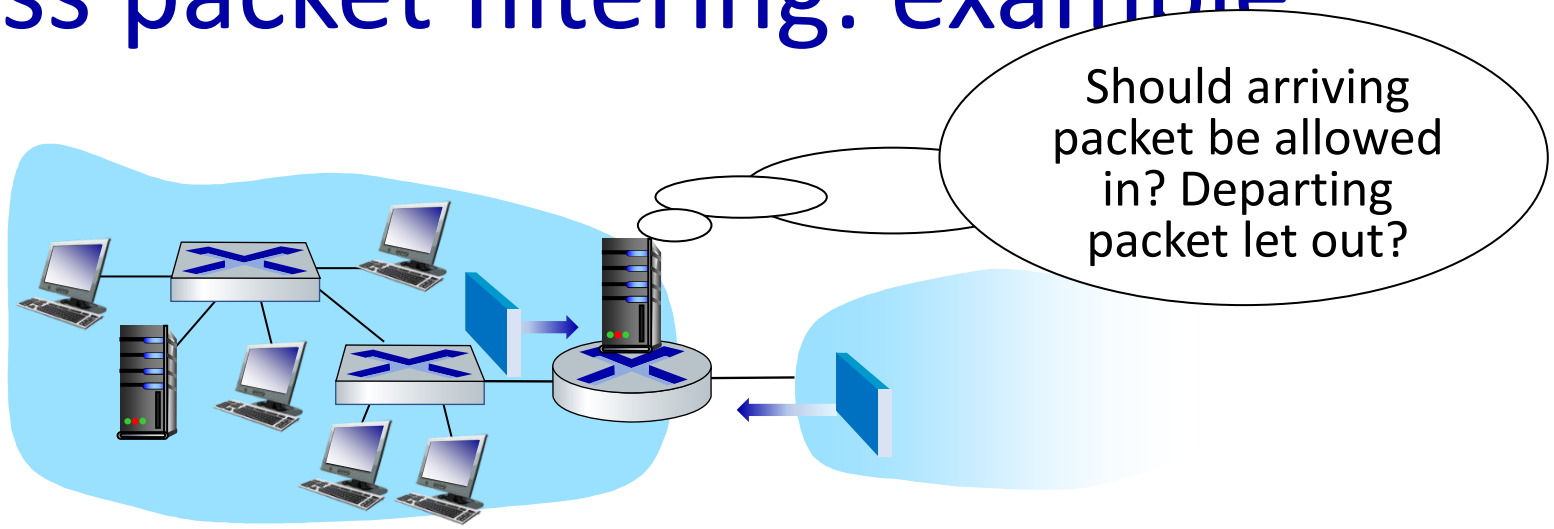
- stateless packet filters
- stateful packet filters
- application gateways

# Stateless packet filtering




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

| 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 (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 (Ch. 4)!

| 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): determine whether incoming, outgoing packets "makes sense"
  - timeout inactive connections at firewall: no longer admit packets

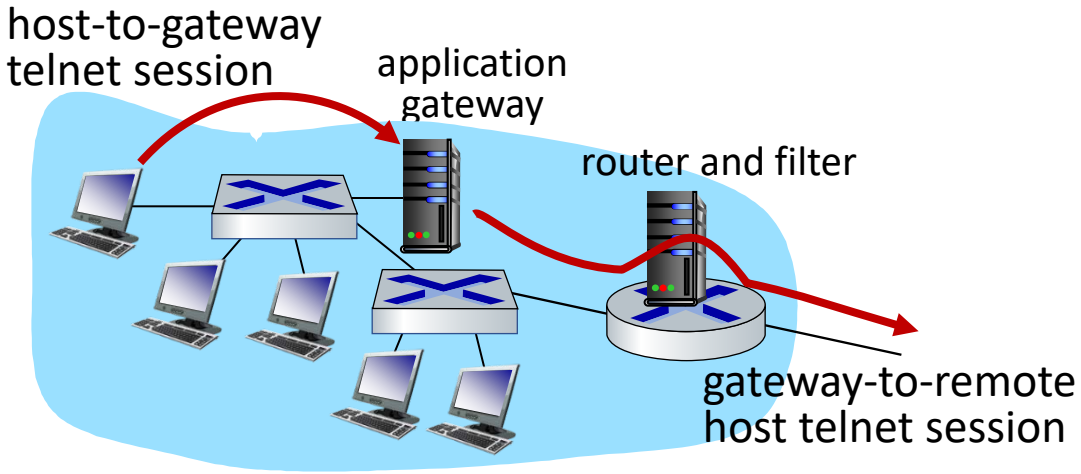
# Stateful packet filtering

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 connection |
|---|---|---|---|---|---|---|---|
| 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 apps 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

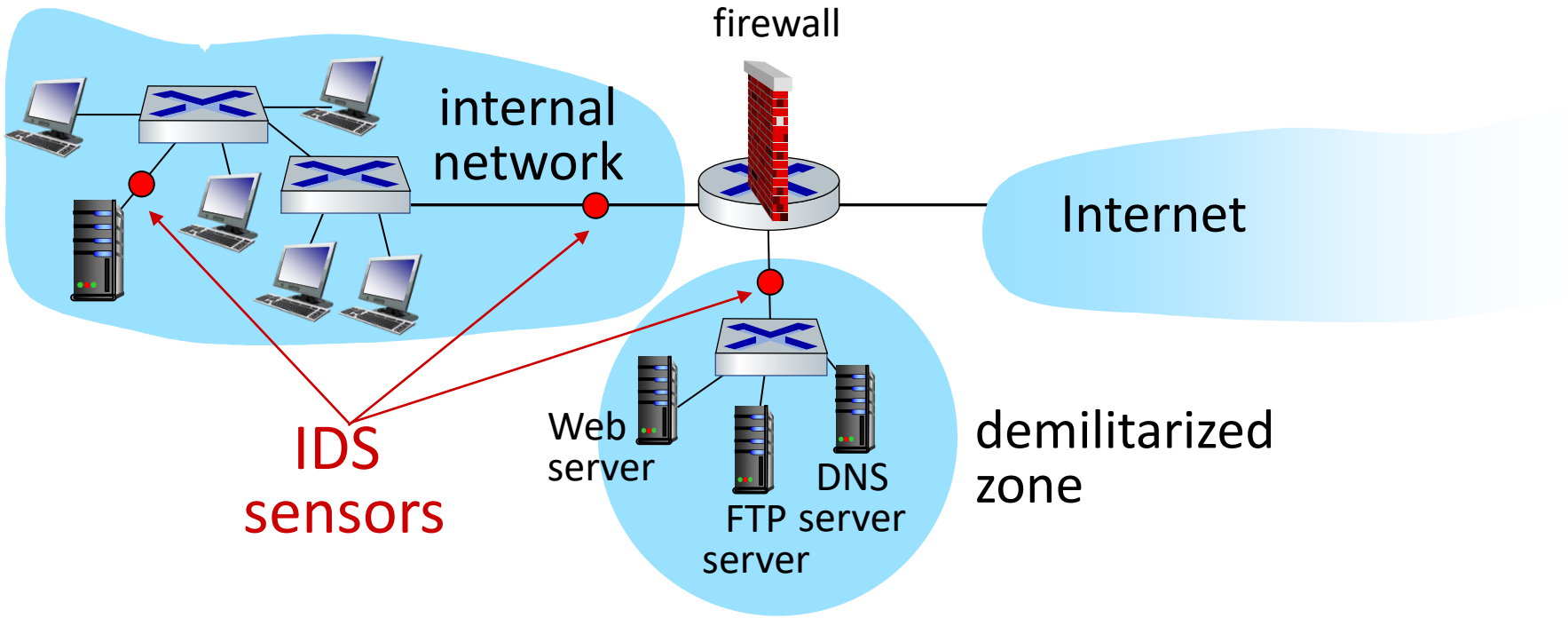
- filters often use all or nothing policy for UDP
- *tradeoff:* degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks

# 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 key)
- message integrity
- end-point authentication

…. used in many different security scenarios

- secure email
- secure transport (TLS)
- IP sec
- 802.11, 4G/5G

operational security: firewalls and IDS