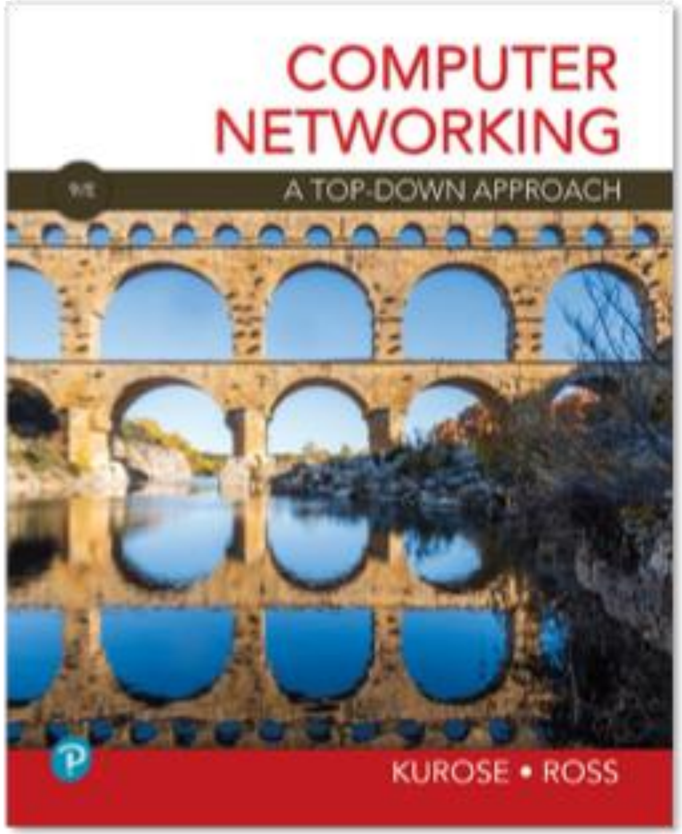
Chapter 8
Security

A note on the use of these PowerPoint slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top-Down Approach
9^{th} edition
Jim Kurose, Keith Ross
Pearson, 2025

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 8 outline

- What is network security?
- Principles of cryptography
- Authentication, digital signatures, message integrity, shared key agreement
- 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

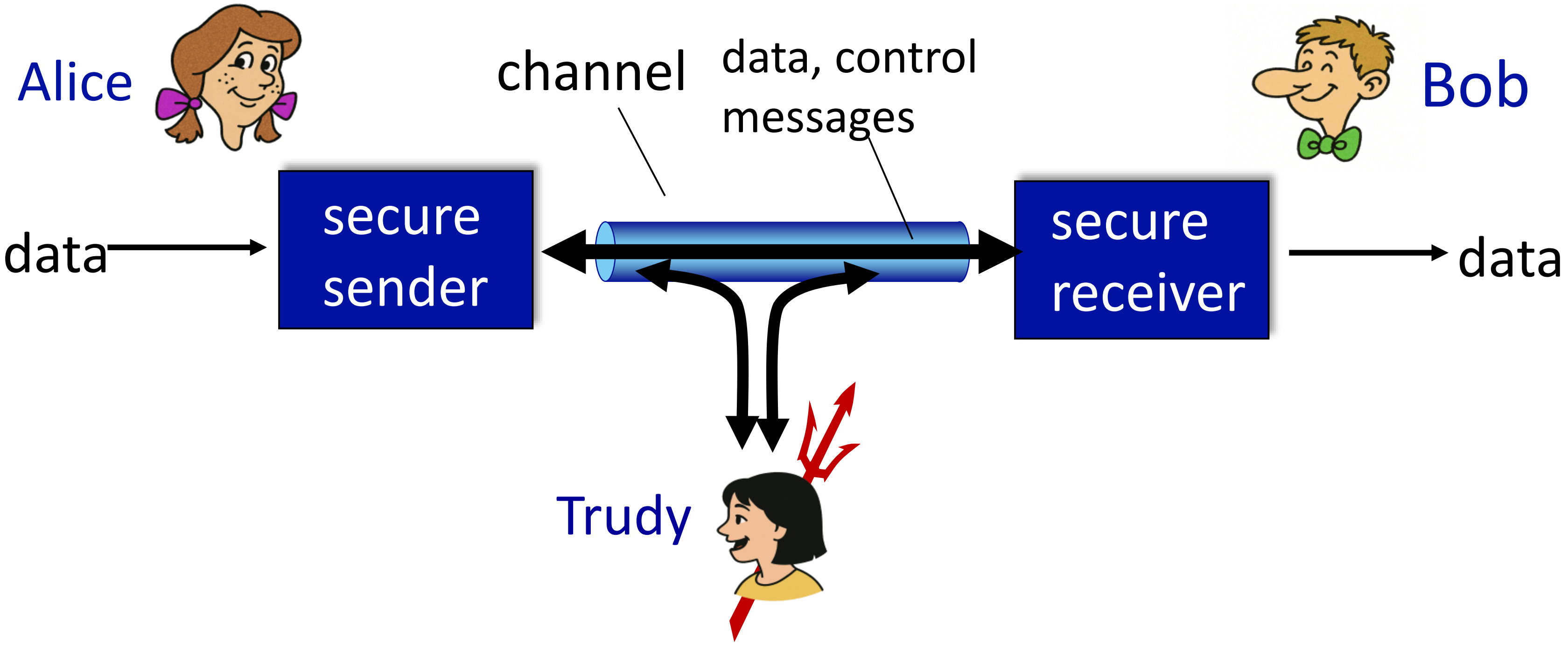
authentication: sender, receiver want to confirm identity of each other

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?

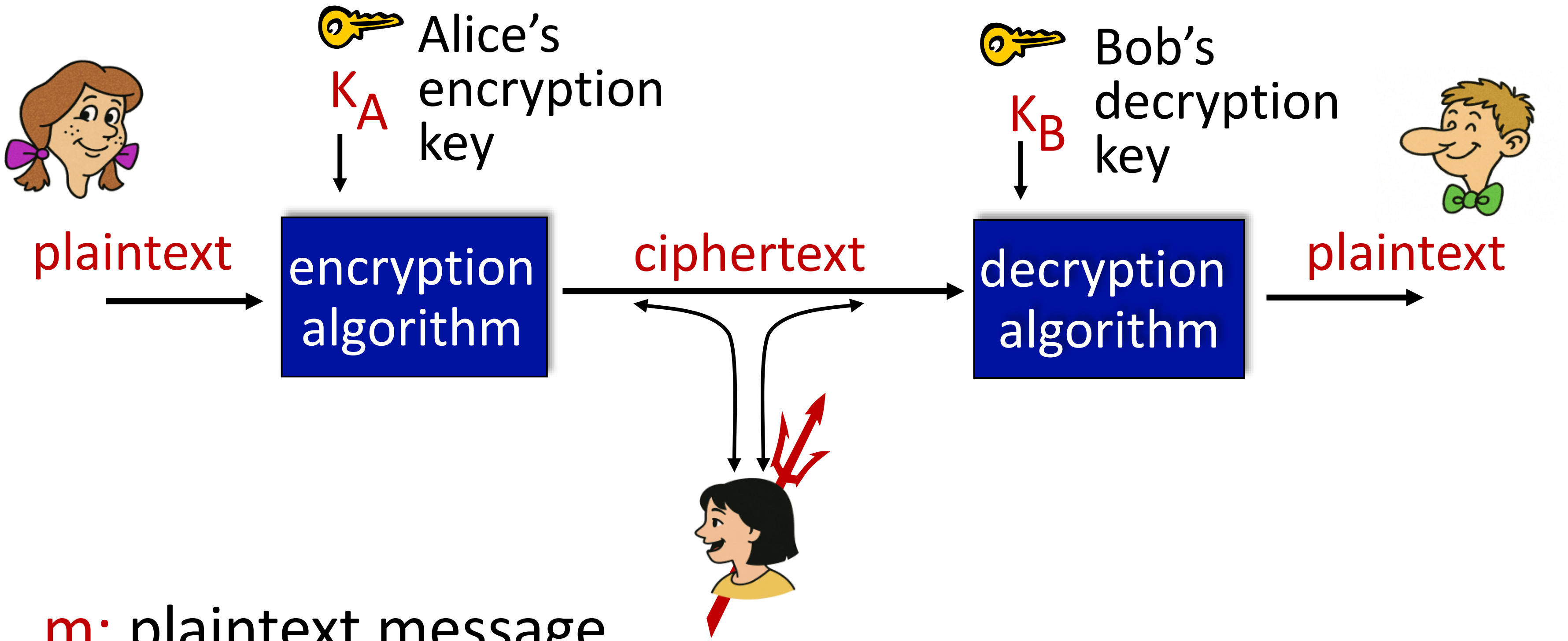
A: A lot! (recall section 1.6)

- 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 8 outline

- What is network security?
- Principles of cryptography
- Authentication, digital signatures, message integrity, shared key agreement
- 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

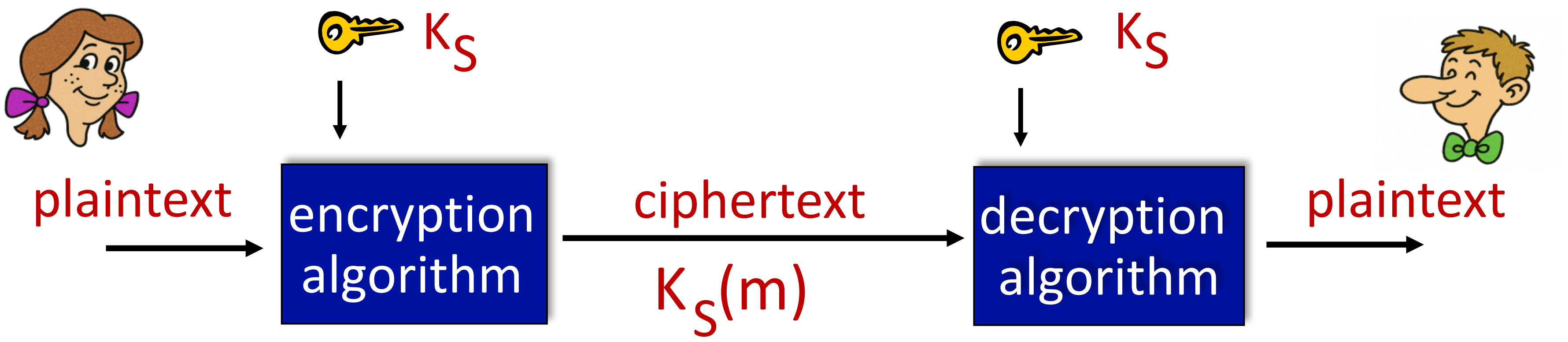
$K_A(m)$: ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

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

```
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

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

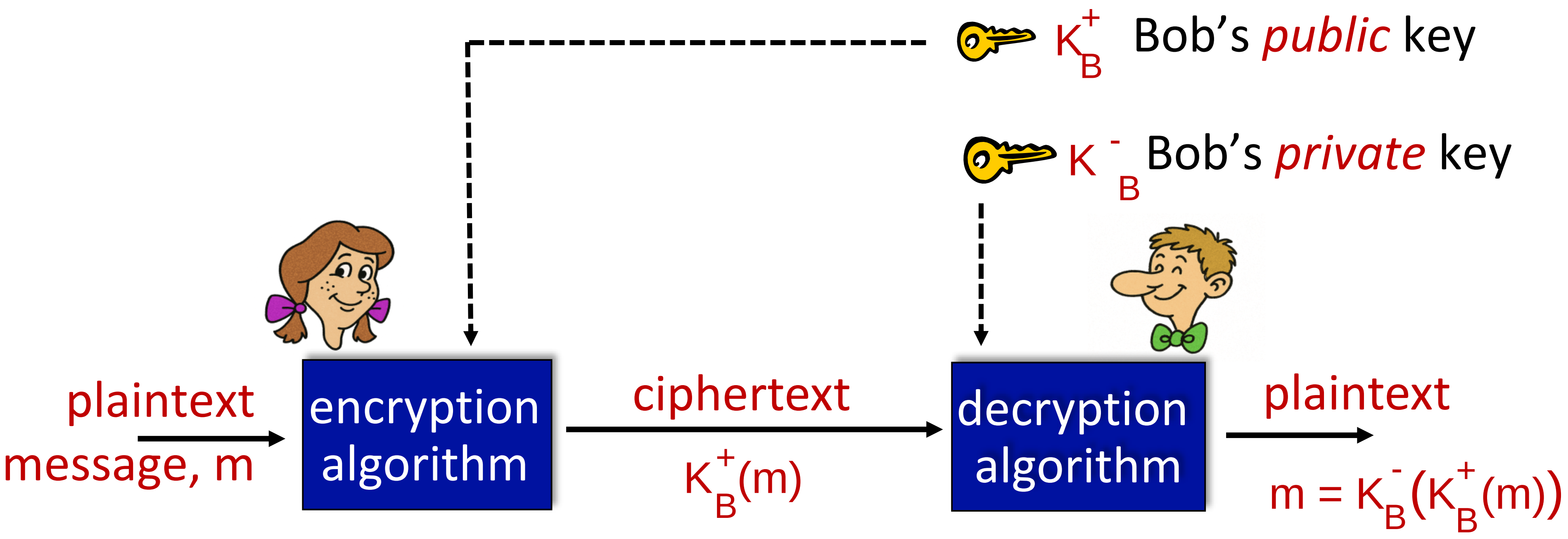
symmetric key crypto:

- 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 $n = pq$, $z = (p-1)(q-1)$

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 $\underbrace{(n,e)}_{K_B^+}$. *private* key is $\underbrace{(n,d)}_{K_B^-}$.

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

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.

	bit pattern	m	m^e	$c = m^e \bmod n$
encrypt:	00001000	12	24832	17

	c	c^d	$m = c^d \bmod n$
decrypt:	17	481968572106750915091411825223071697	12

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,

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

RSA: another important property

The following property will be *very* useful later:

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

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

Using Public-Key Cryptography

We'll (next) see *lots* of non-confidentiality uses of public-key cryptography!

- authentication
- digital signatures
- message integrity
- symmetric-key agreement

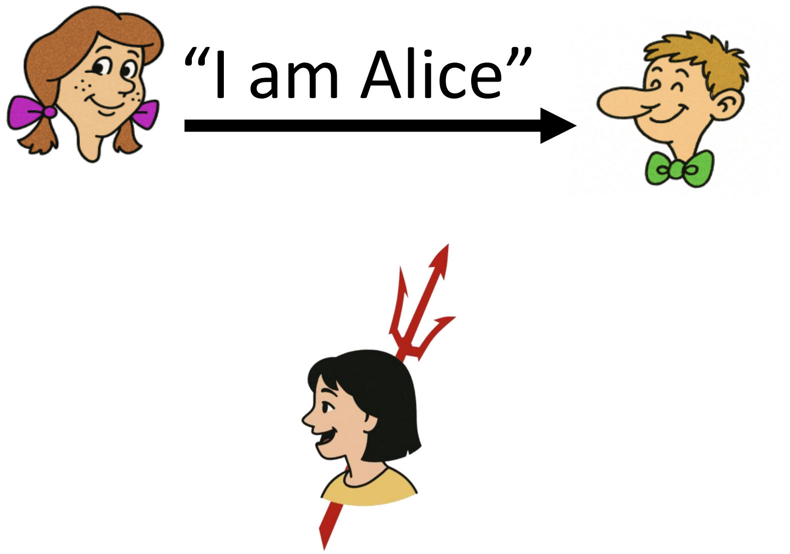
Chapter 8 outline

- What is network security?
- Principles of cryptography
- Use of public-key cryptography: authentication, digital signatures, message integrity, shared key agreement
- 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"

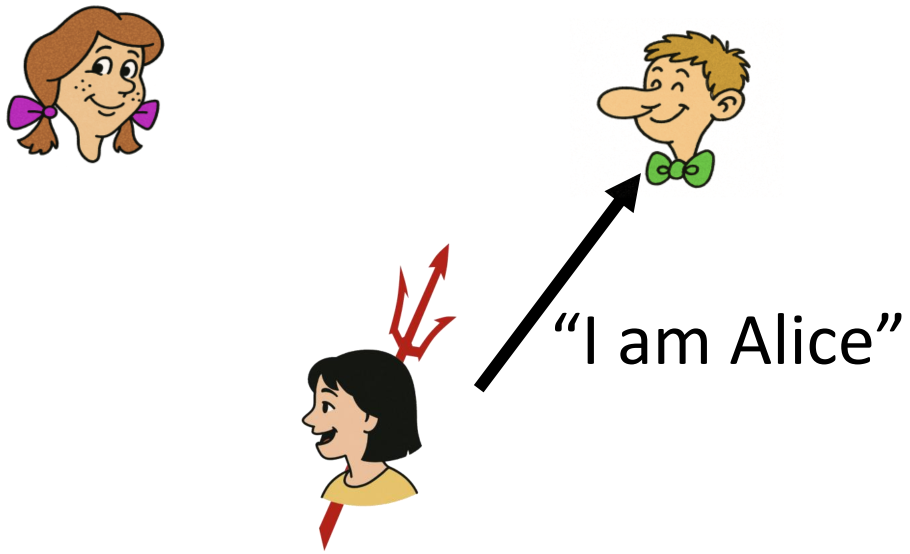


failure scenario??

Authentication

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

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

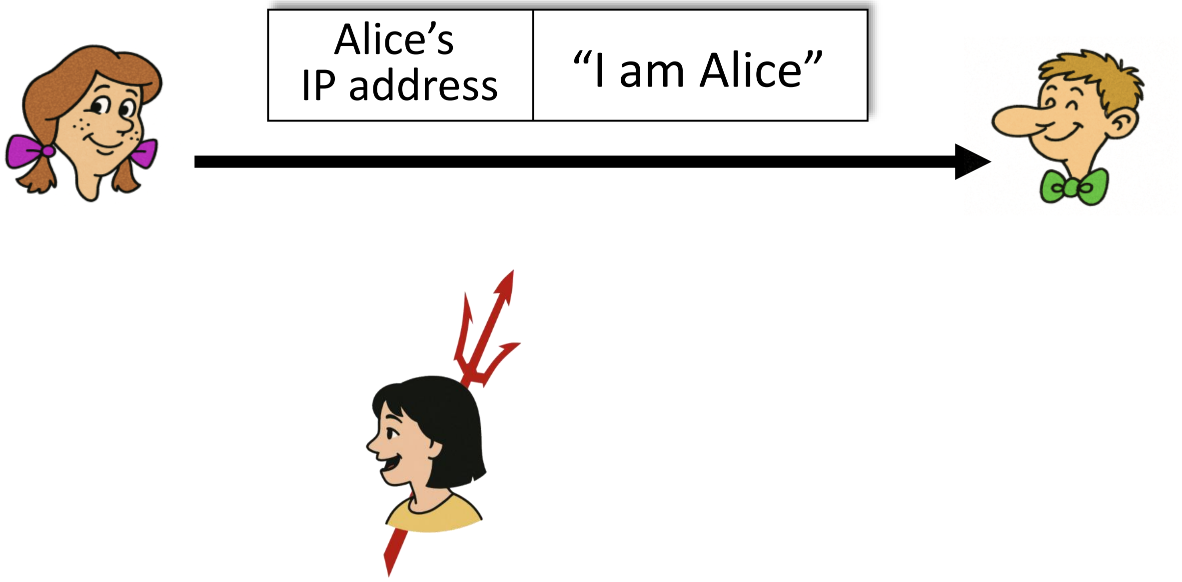


in a network, Bob can not "see" Alice, so Trudy simply declares herself to be Alice

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

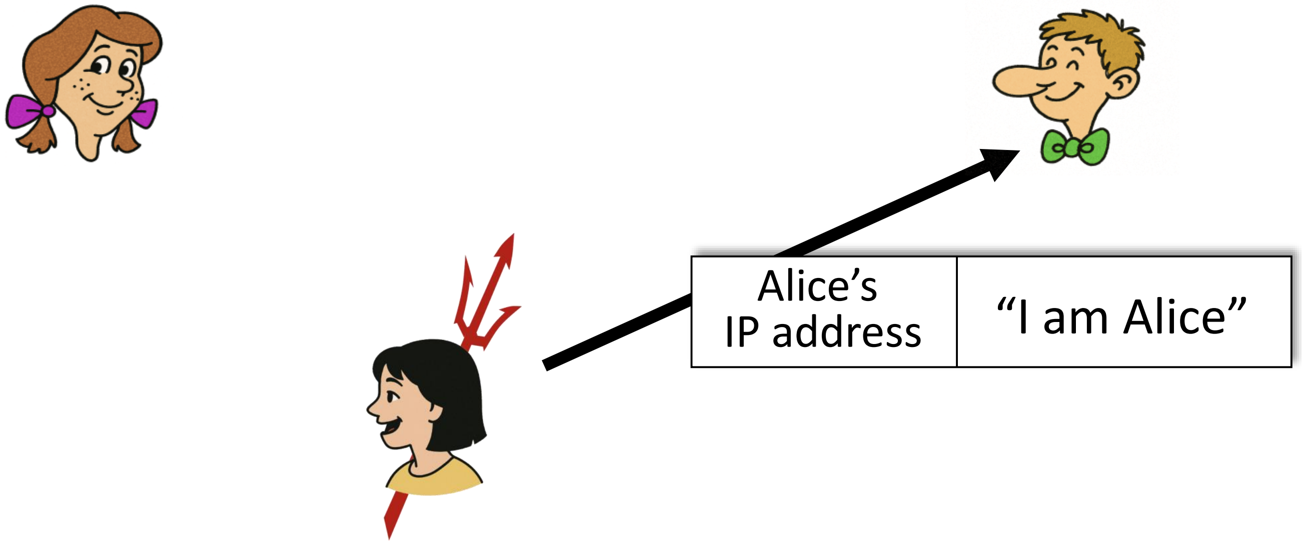


failure scenario??

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

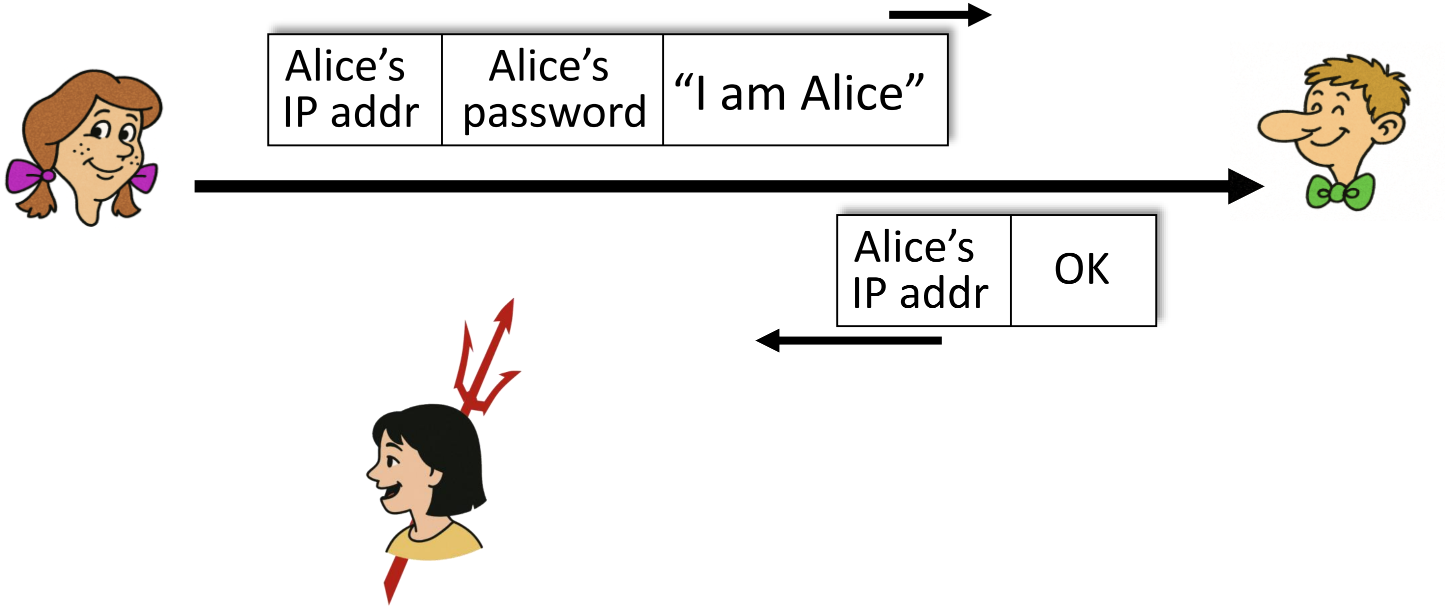


Trudy can create a packet “spoofing” Alice’s address

Authentication: a third try

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

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

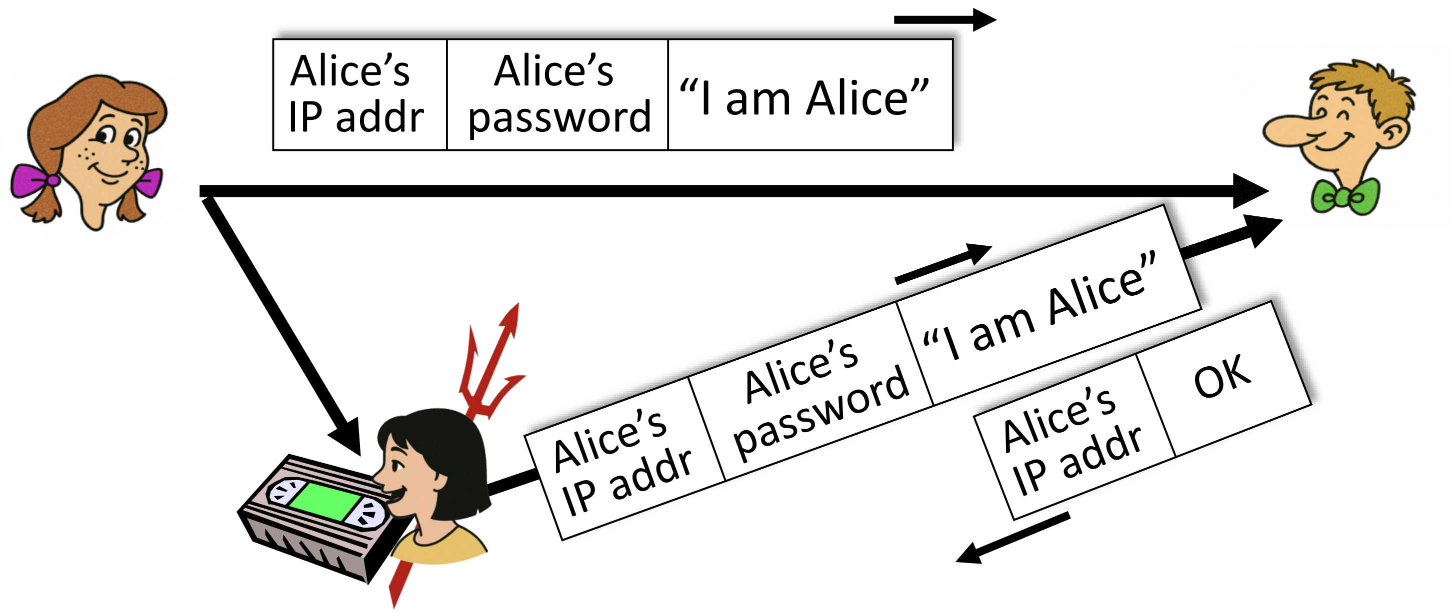


failure scenario??

Authentication: a third try

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

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

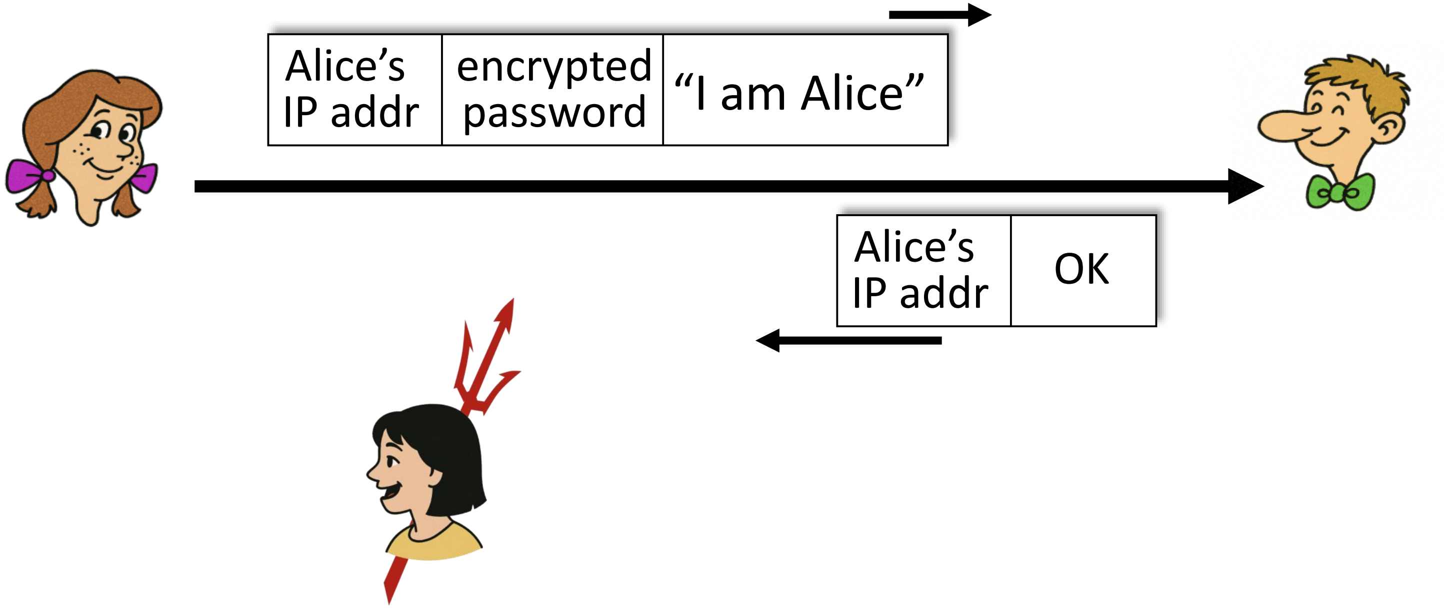


playback attack: *Trudy records Alice's packet and later plays it back to Bob*

Authentication: a modified third try

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

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

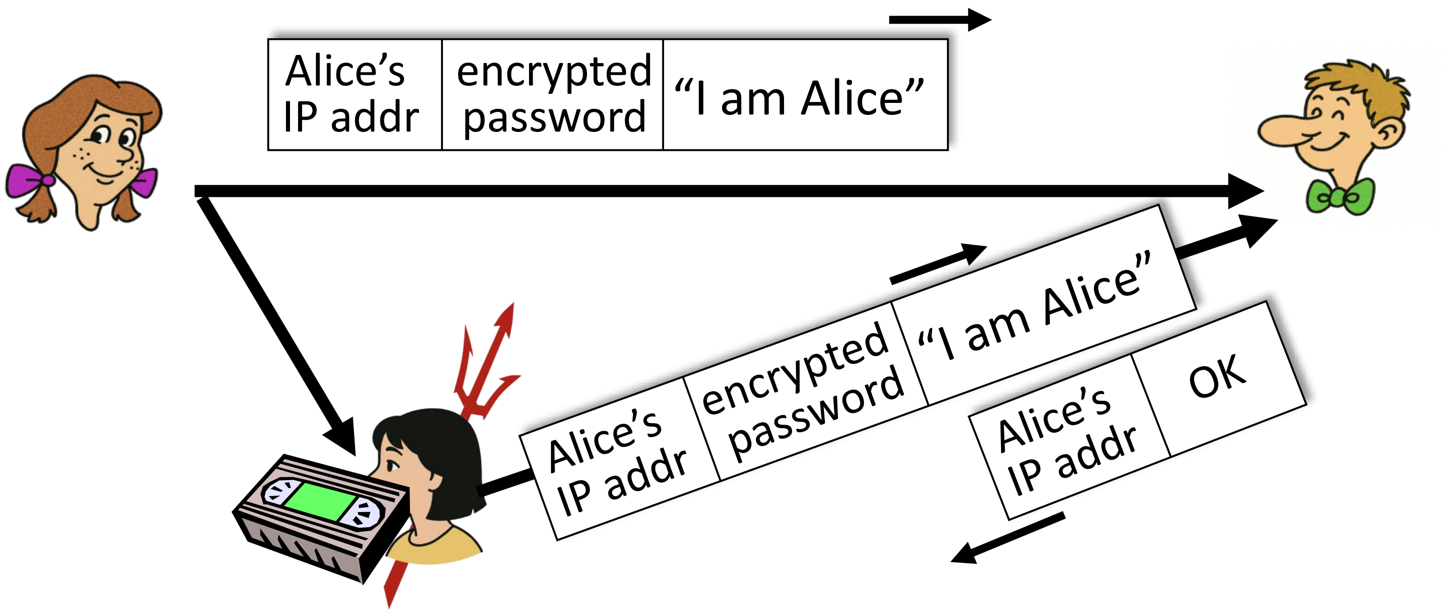


failure scenario??

Authentication: a modified third try

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

Protocol ap3.0: Alice says "I am Alice" 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

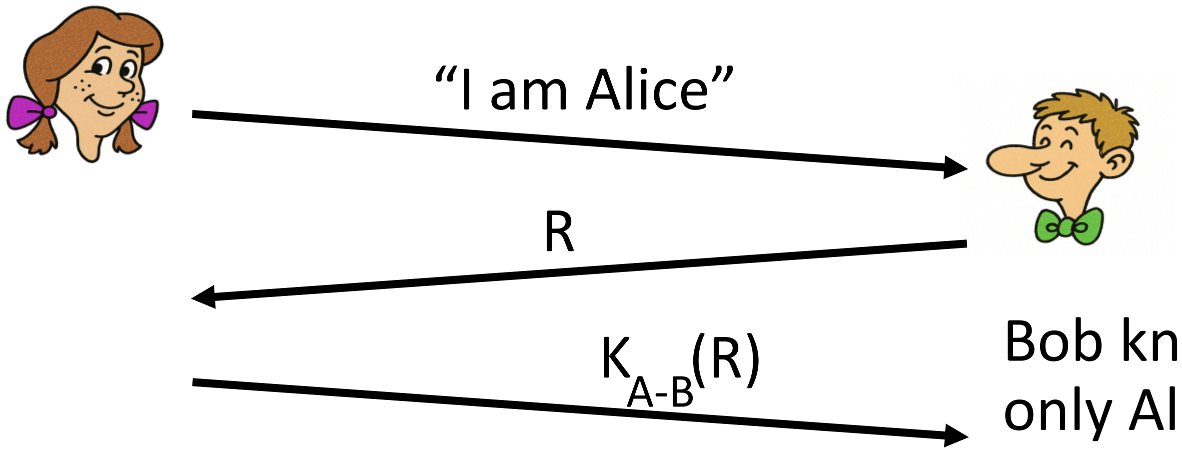
Authentication: a fourth try

Goal: avoid playback attack

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

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

- Alice must return R, encrypted with shared secret key



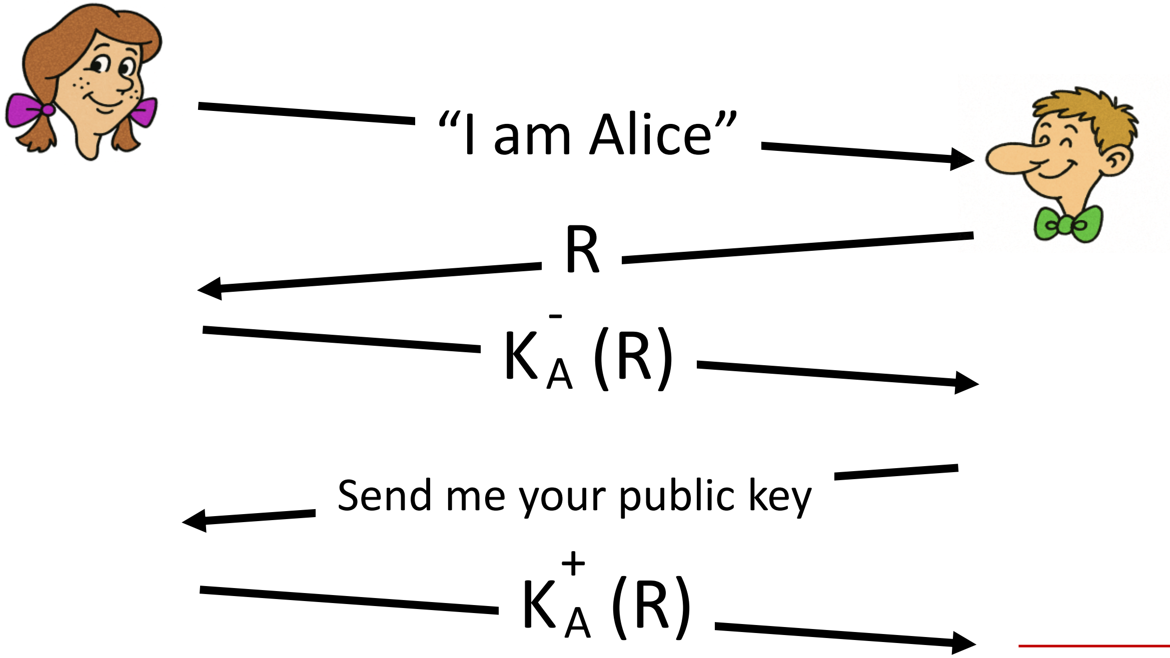
Bob know Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

Failures, drawbacks?

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

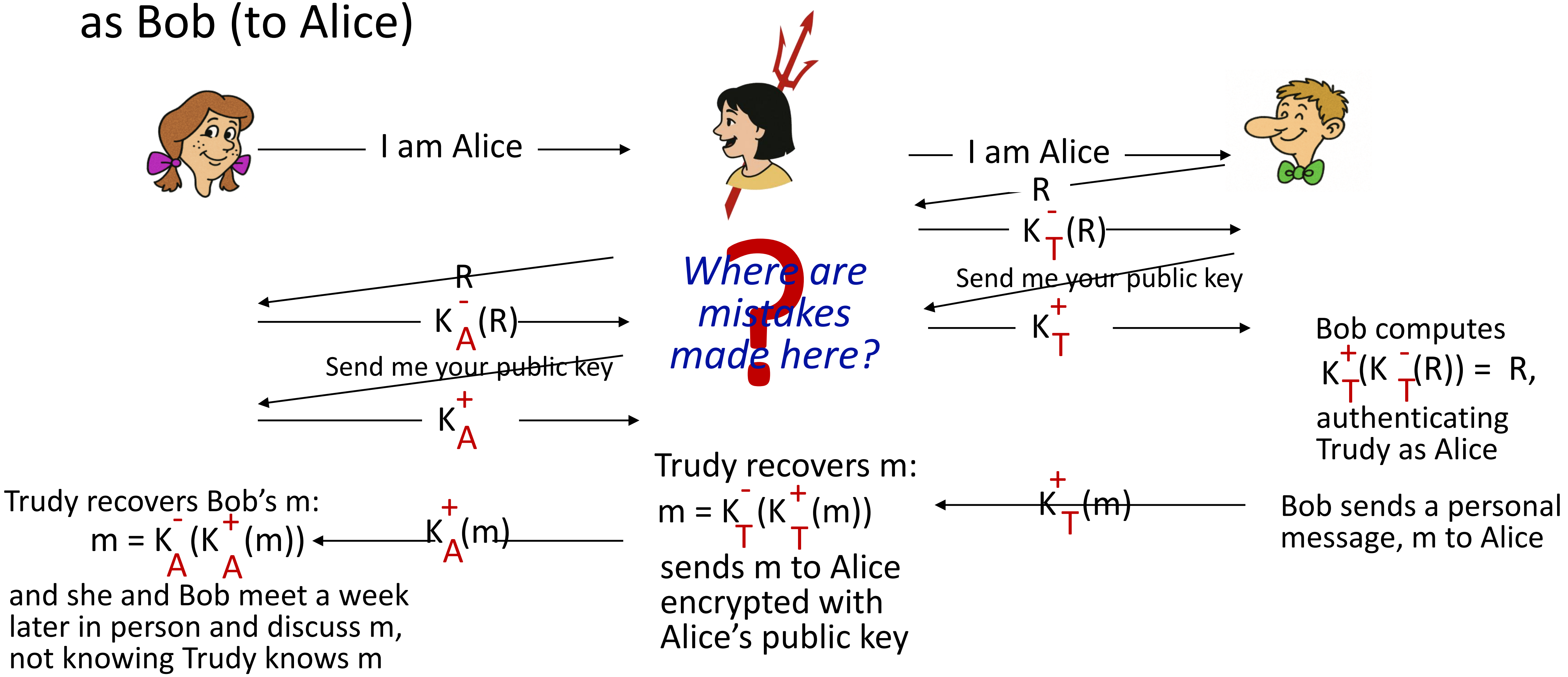
$$K_A^+(K_A^-(R)) = R$$

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

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

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)



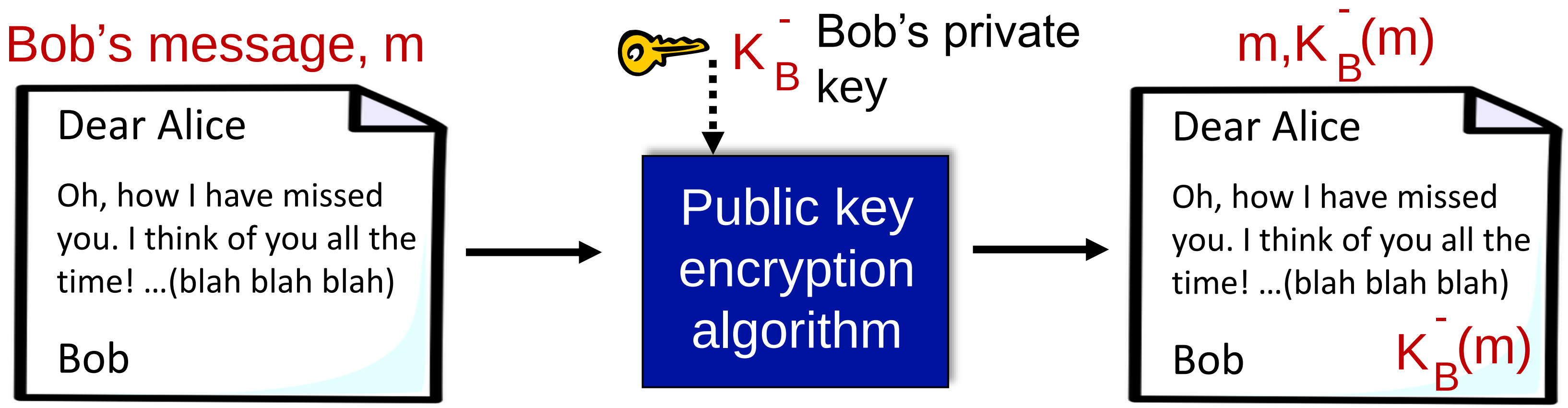
Chapter 8 outline

- What is network security?
- Principles of cryptography
- Use of public-key cryptography: authentication, digital signatures, message integrity, shared key agreement
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

Digital signatures

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 $K_B^+(K_B^-(m)) = m$, 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:

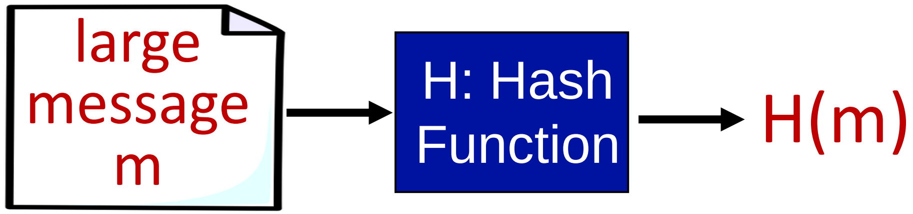
✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

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

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

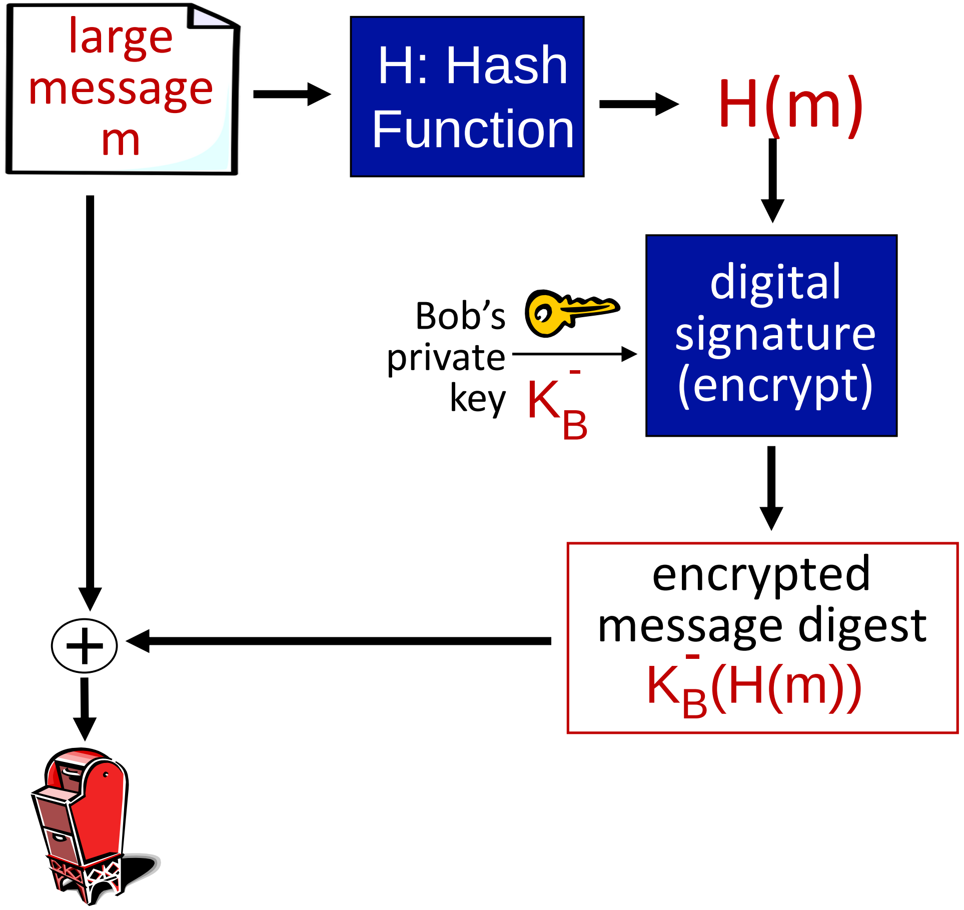
message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42
	B2 C1 D2 AC

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

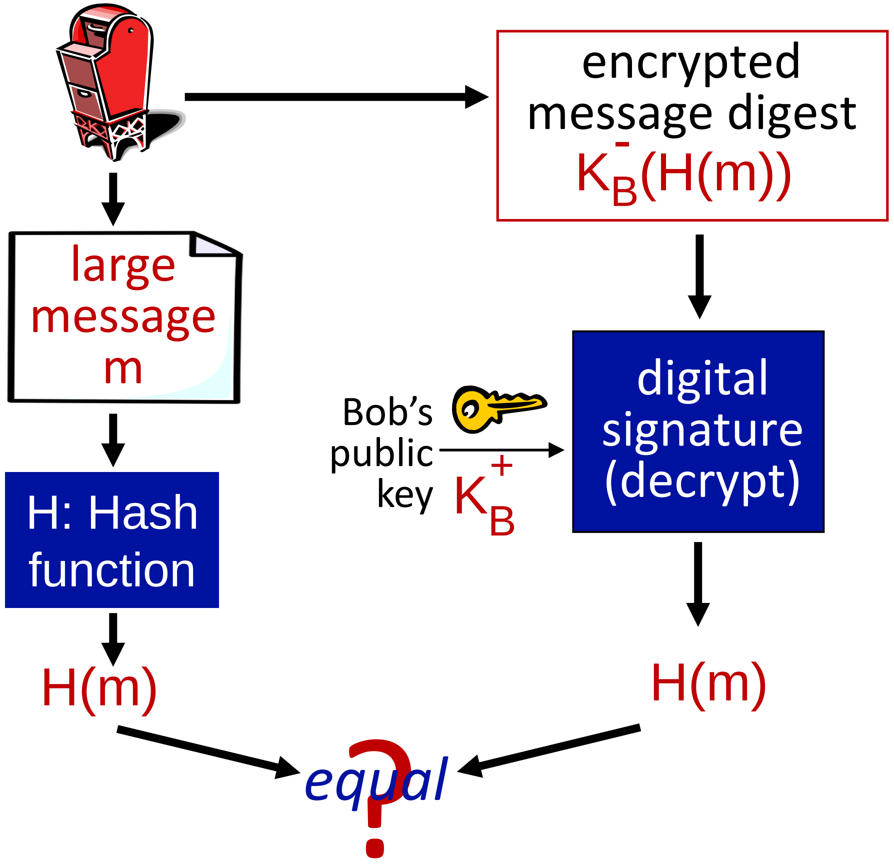
different messages but identical checksums!

Digital signature = signed message digest

Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:

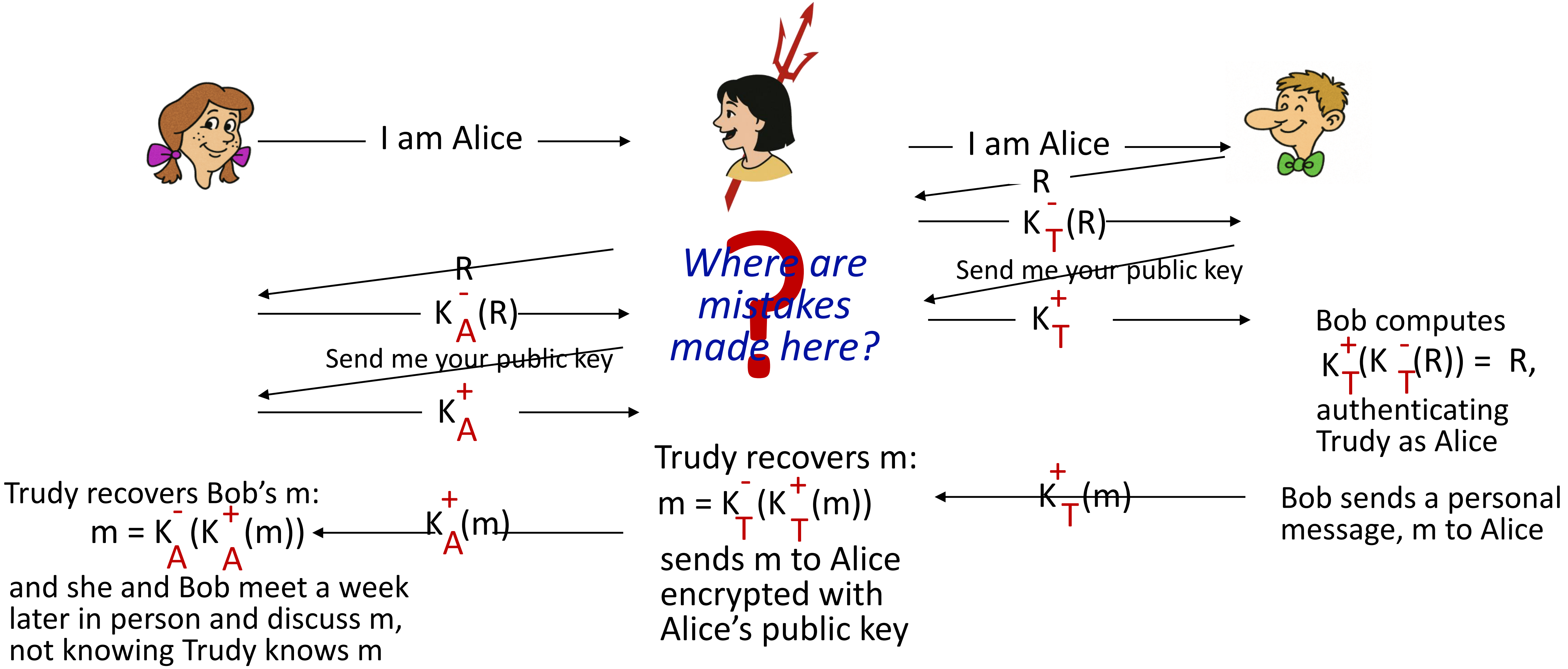


Hash function algorithms

- MD5 hash function widely used (RFC 1321)
  - 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
  - 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)

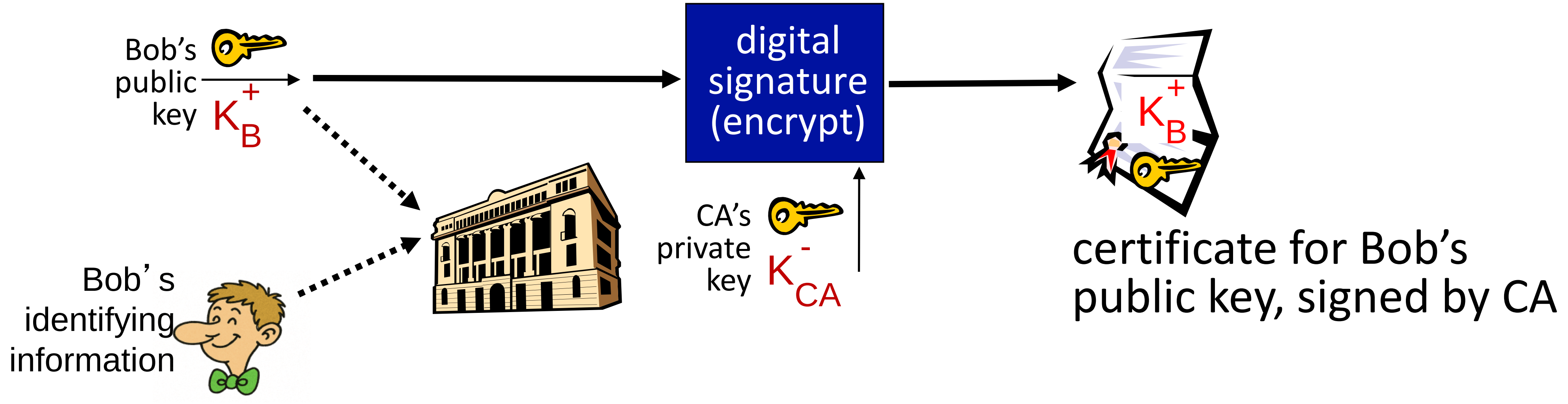


Need for certified public keys

- 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

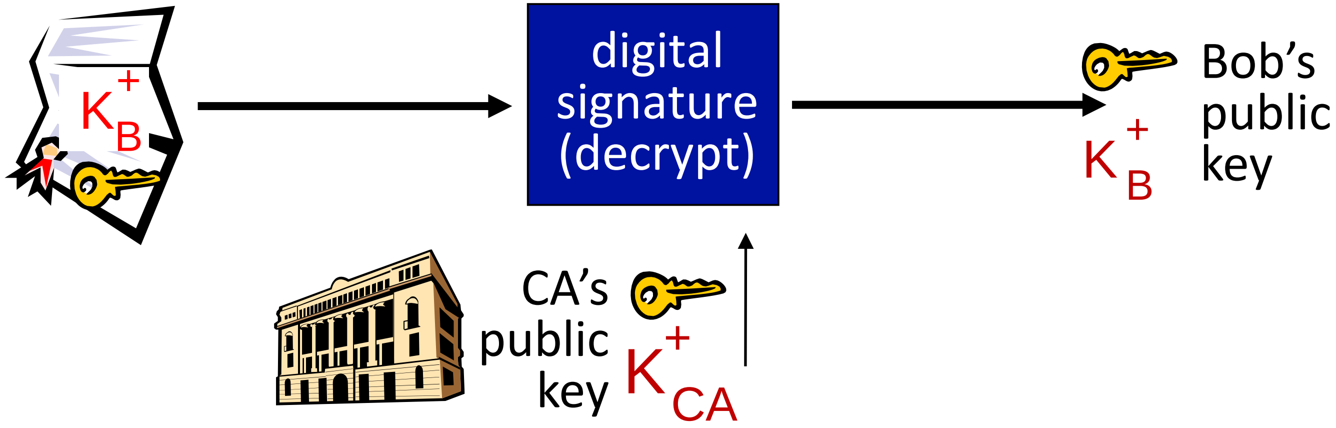
Public key Certification Authorities (CA)

- 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 8 outline

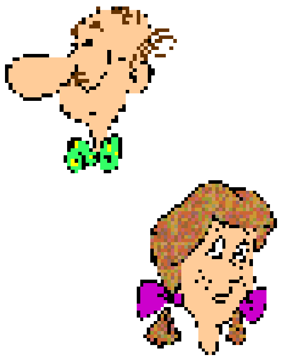
- What is network security?
- Principles of cryptography
- Use of public-key cryptography: authentication, digital signatures, message integrity, shared key agreement
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Security in wireless and mobile networks
- Operational security: firewalls and IDS

Using Public-Key Cryptography to Agree on a Shared Key

Problem: for Bob and Alice to use symmetric key techniques they must agree on the shared key.

- What if they've never met before?
- *public-key cryptography can solve this problem too!*

Diffie-Hellman (DH) public/private keys: different algorithm than RSA



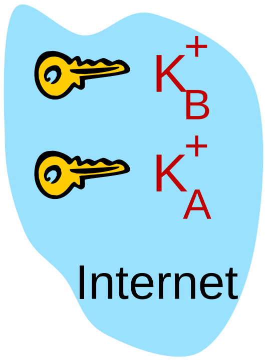
- Bob chooses *public* values: large prime number *p,* another large number *g (<p).*
- Alice, Bob independently choose private values: *SA, SB*
- Alice's public key: $K_A^+ = g^{SA} \bmod p$
- Bob's public key: $K_B^+ = g^{SB} \bmod p$

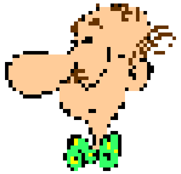
Using Public-Key Cryptography to Agree on a Shared Key

Diffie-Hellman (DH) shared key derivation:

- Alice, Bob compute a common shared secret without ever explicitly exchanging this shared secret!
- Alice, Bob have DH public keys: K_A^+ , K_B^+ , secret values SA, SB

SA





SB

Alice's computed shared secret (uses Bob's public key and SA):

$(K_B^+)^{SA} \bmod p$

Bob's computed shared secret (uses Alice's public key and SB):

$(K_A^+)^{SB} \bmod p$

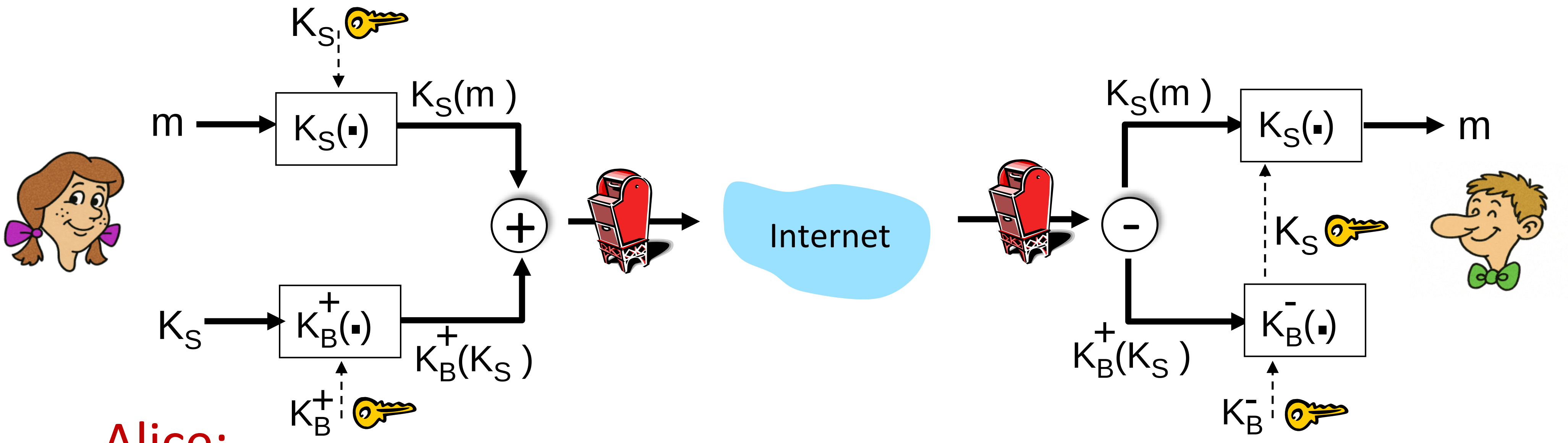
Alice's and Bob's locally computed values are equal to each other!

Chapter 8 outline

- What is network security?
- Principles of cryptography
- Authentication, digital signatures, message integrity, shared key agreement
- 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.

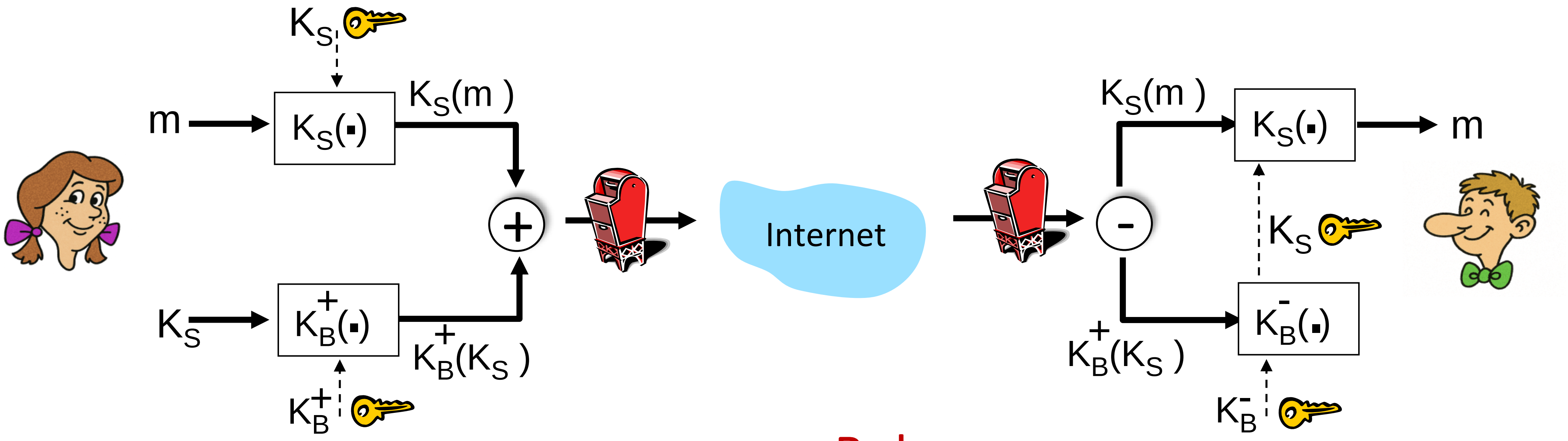


Alice:

- 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.

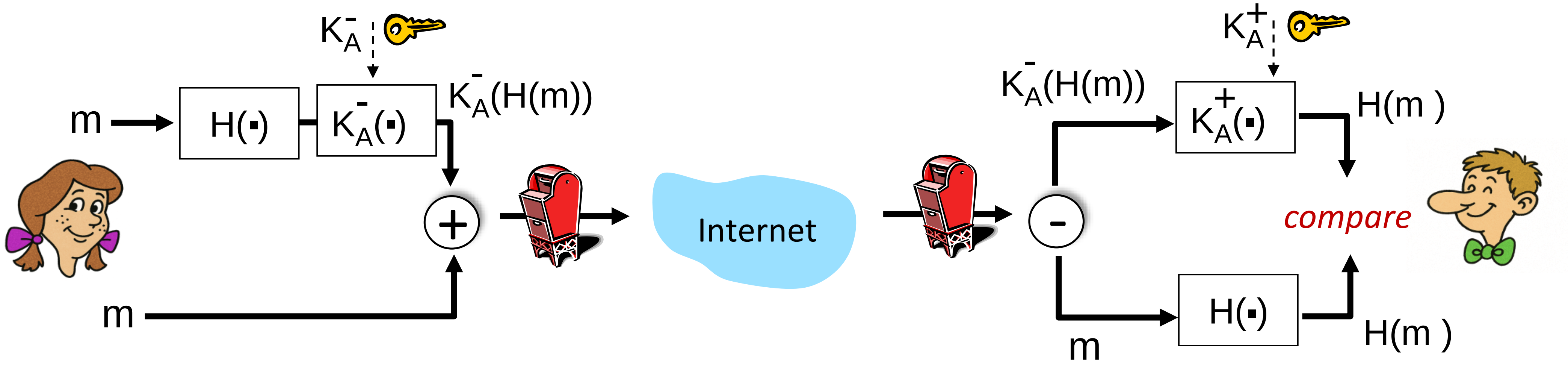


Bob:

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

Secure e-mail: integrity, authentication

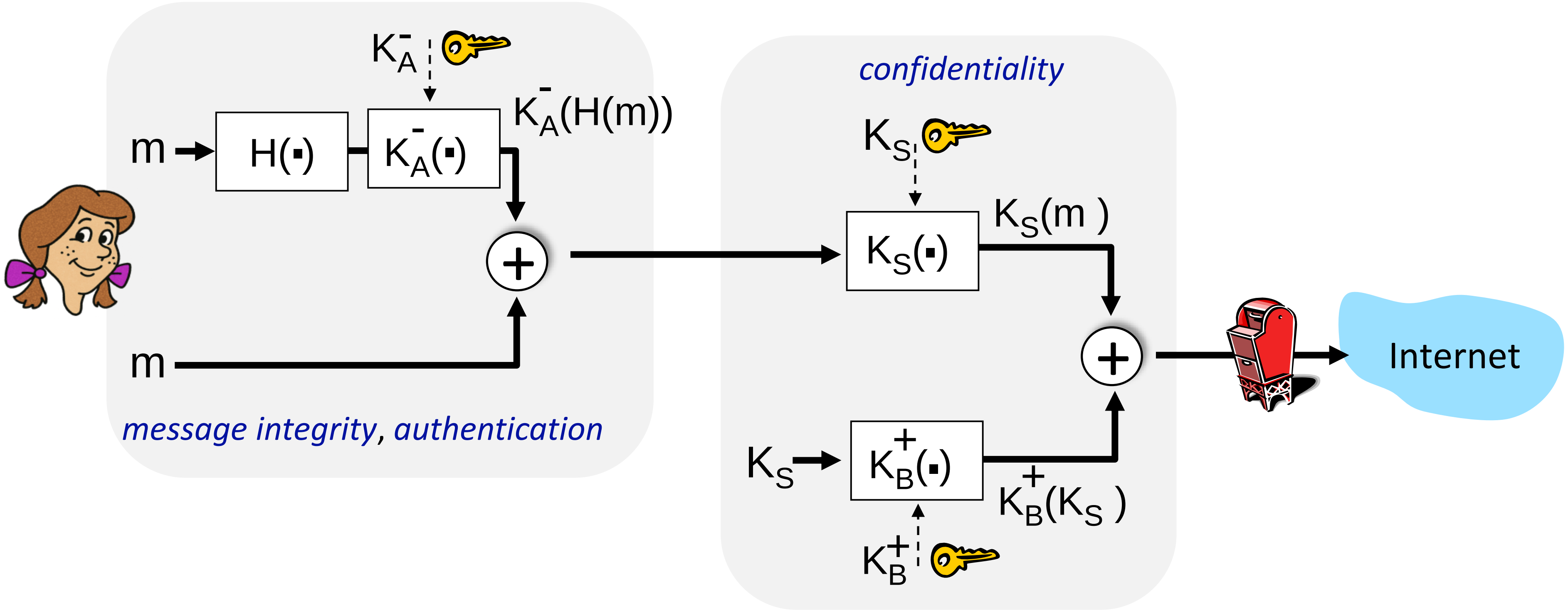
Alice wants to send m to Bob, with *message integrity*, *authentication*



- 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*



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

What are Bob's complementary actions?

Chapter 8 outline

- What is network security?
- Principles of cryptography
- Authentication, digital signatures, message integrity, shared key agreement
- 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)

- 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*

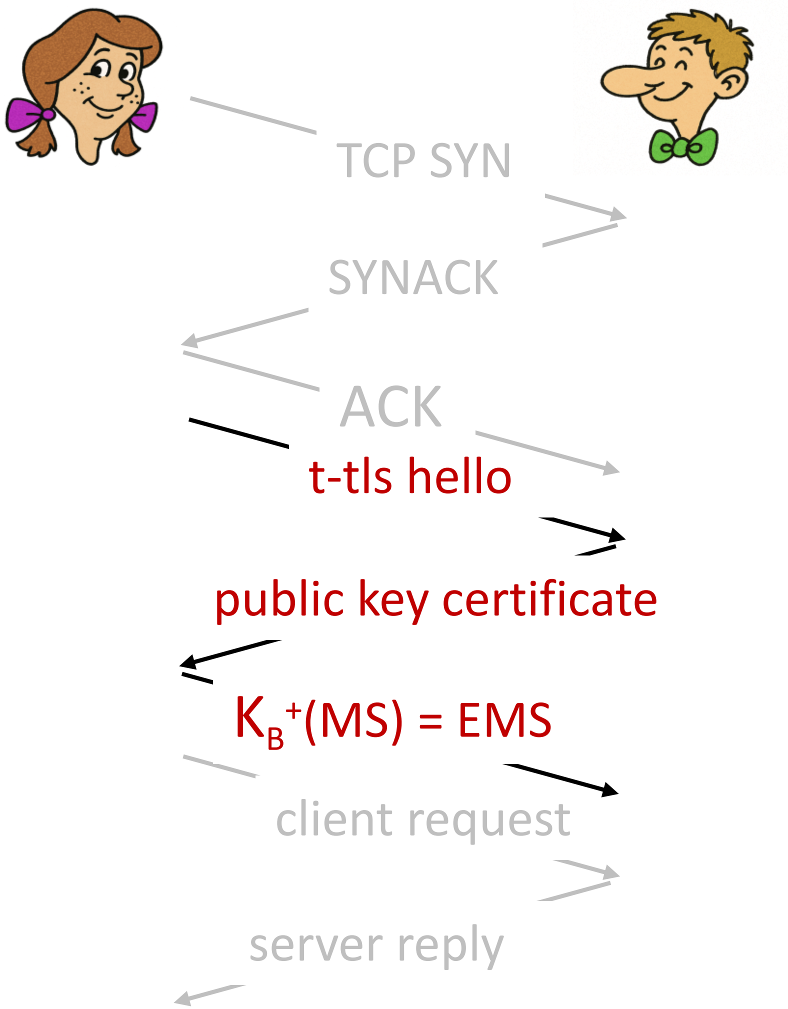
all techniques we have studied!

- 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
  - connection closure: special messages to securely close connection

t-tls: initial handshake



t-tls handshake phase:

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

t-tls: cryptographic keys

- considered bad to use same key for more than one cryptographic function
  - different keys for message authentication code (MAC) and encryption
- four keys:
  - K_c : encryption key for data sent from client to server
  - M_c : MAC key for data sent from client to server
  - K_s : encryption key for data sent from server to client
  - M_s : MAC key for data sent from server to client
- 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:

K_c(| *length* | *data* | *MAC* |)

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

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 #

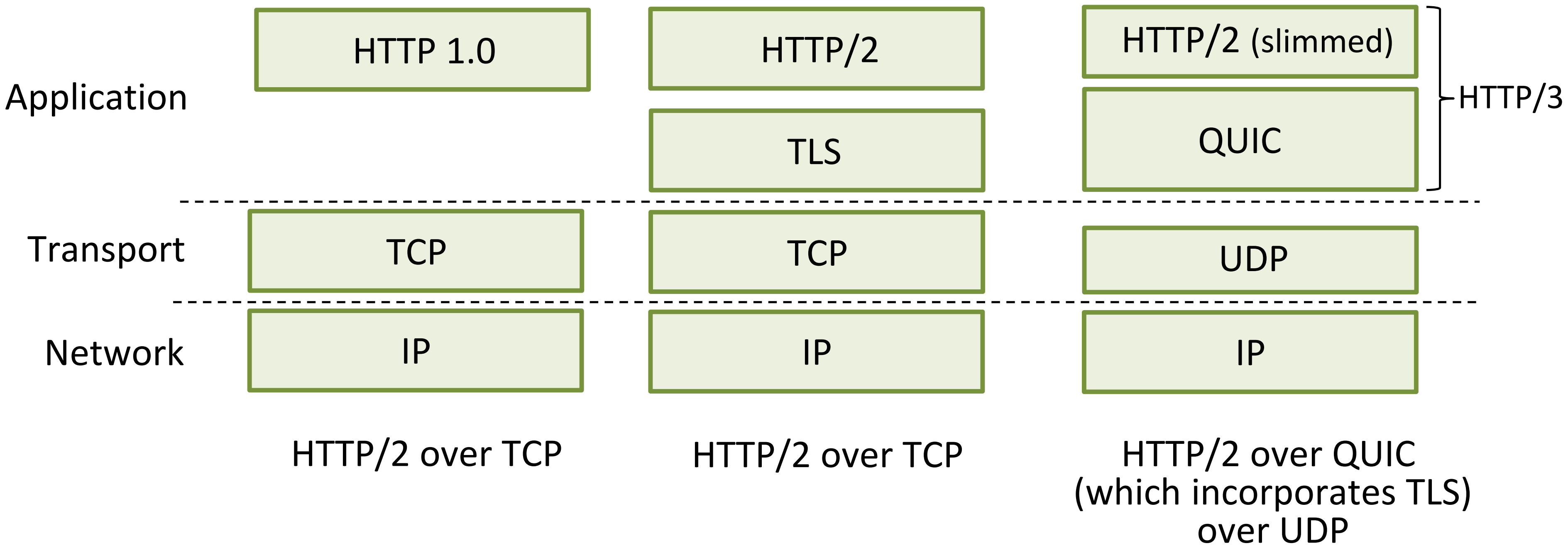
$K_c($

length	*type*	*data*	*MAC*

$)$

Transport-layer security (TLS)

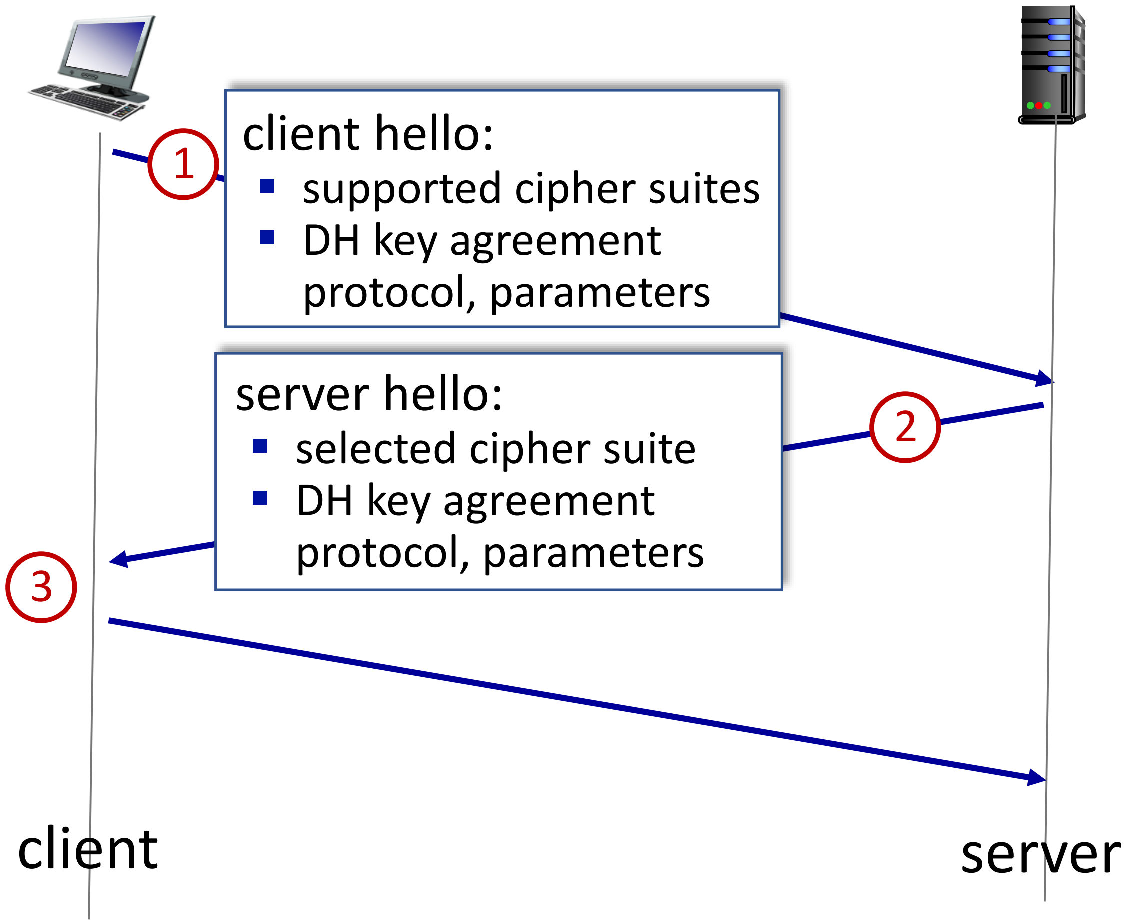
- TLS provides an API that *any* application can use
- an HTTP view of TLS:



TLS: 1.3 cipher suite

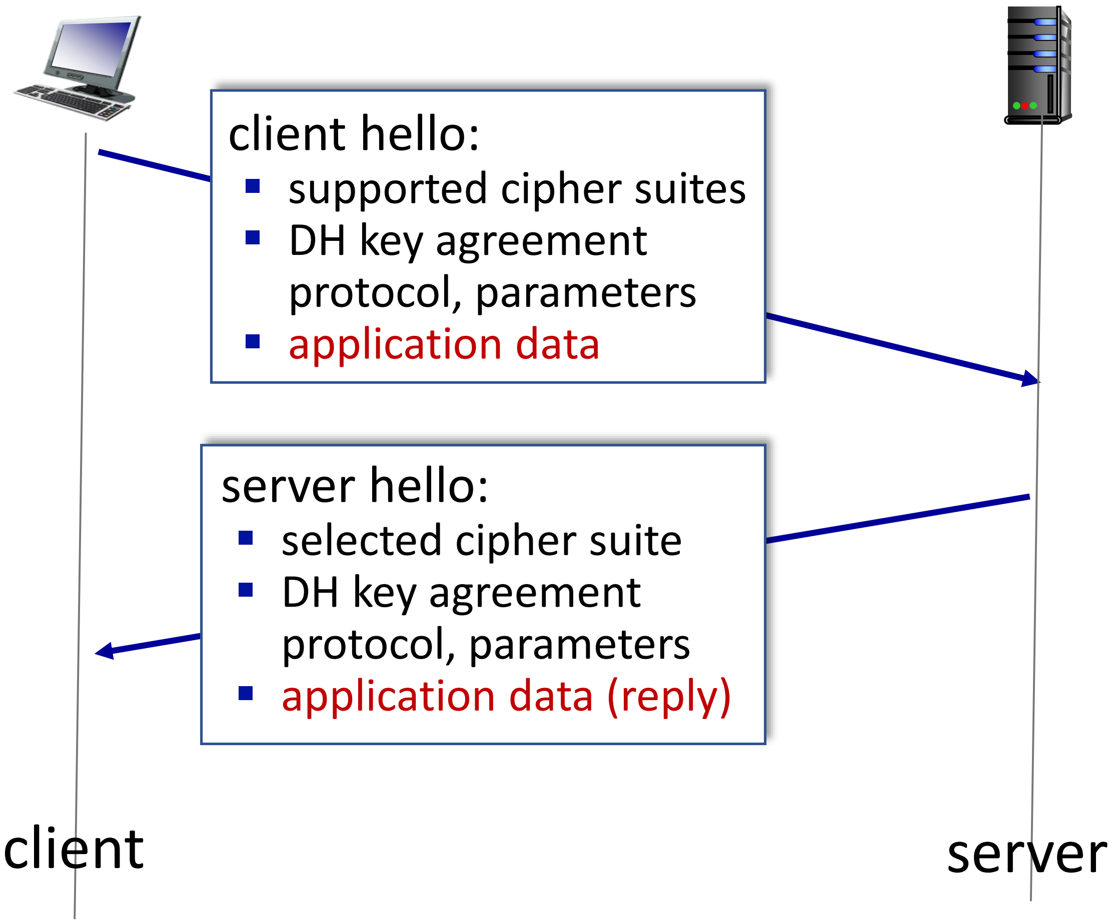
- "cipher suite": algorithms that can be used for key generation, encryption, MAC, digital signature
- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
  - only 5 choices, rather than 37 choices
  - *requires* Diffie-Hellman (DH) for key exchange, rather than DH or RSA
  - combined 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



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



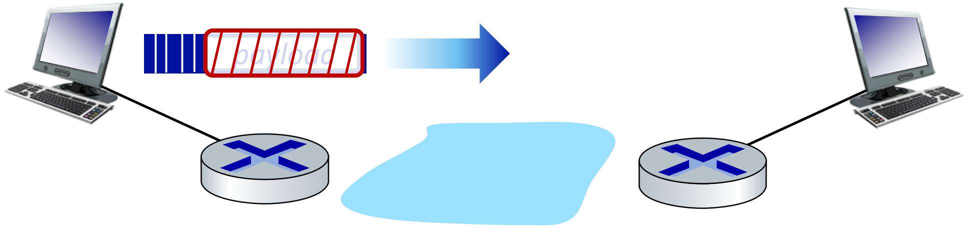
- initial hello message contains encrypted application data!
  - "resuming" earlier connection between client and server
  - application data encrypted using "resumption master secret" from earlier connection
- vulnerable to replay attacks!
  - maybe OK for get HTTP GET or client requests not modifying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

Chapter 8 outline

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

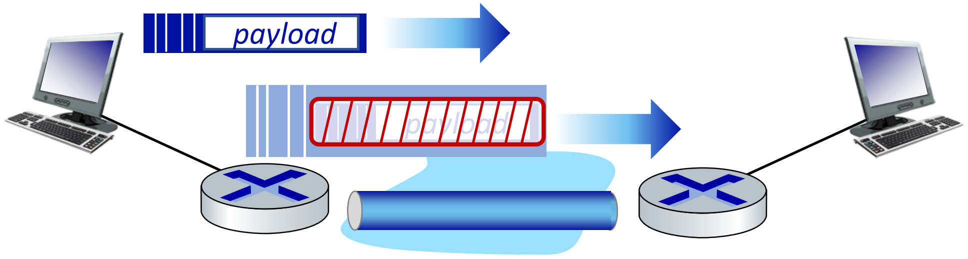
IP Sec

- 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:

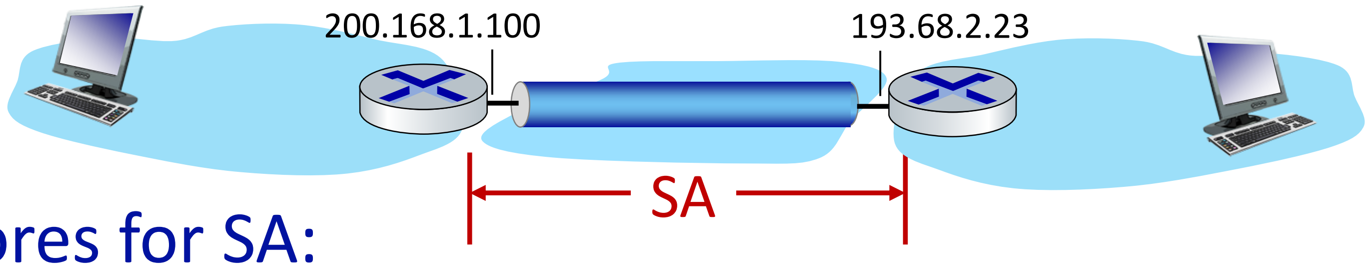
- entire datagram is encrypted, authenticated
- encrypted datagram encapsulated in new datagram with new IP header, tunneled to destination

Two IPsec protocols

- Authentication Header (AH) protocol [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)

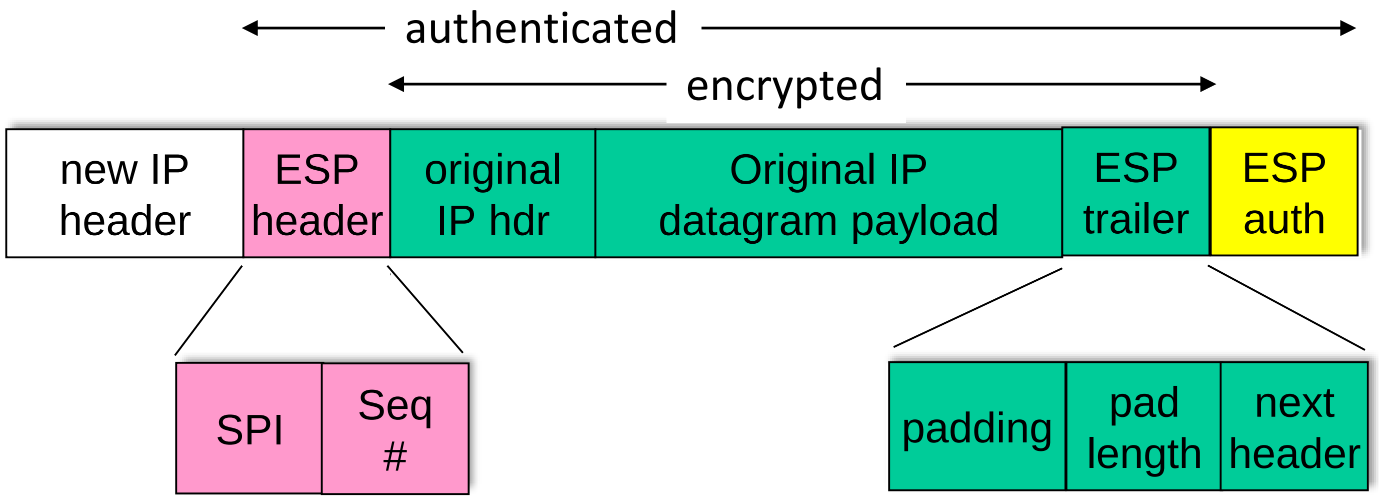
- before sending data, security association (SA) established from sending to receiving entity (directional)
- ending, receiving entitles maintain *state information* about SA
  - recall: TCP endpoints also maintain state info
  - IP is connectionless; IPsec is connection-oriented!



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



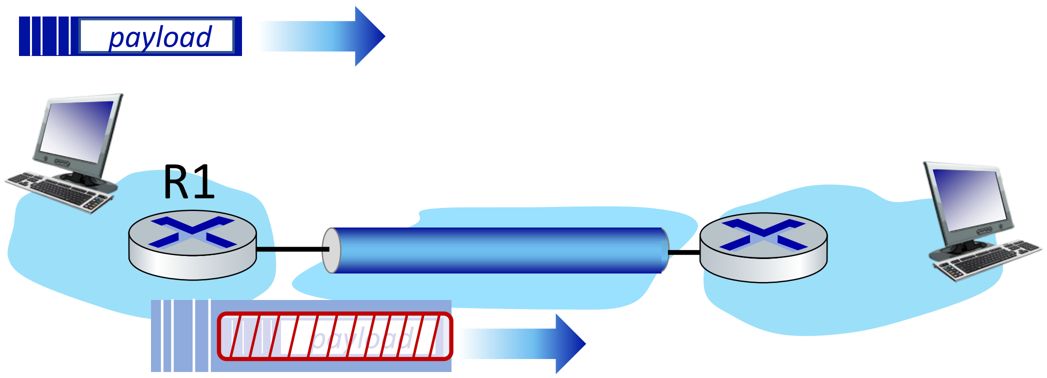
tunnel mode ESP

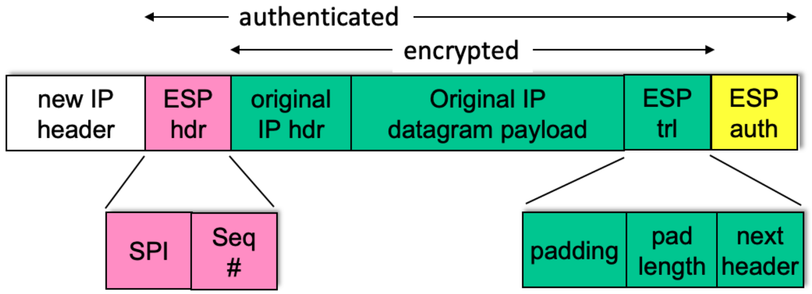
- ESP trailer: padding for block ciphers
- ESP header:
  - SPI, so receiving entity knows what to do
  - sequence number, to thwart replay attacks
- MAC in ESP auth field created with shared secret key

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
  - with PKI (pubic/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.

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

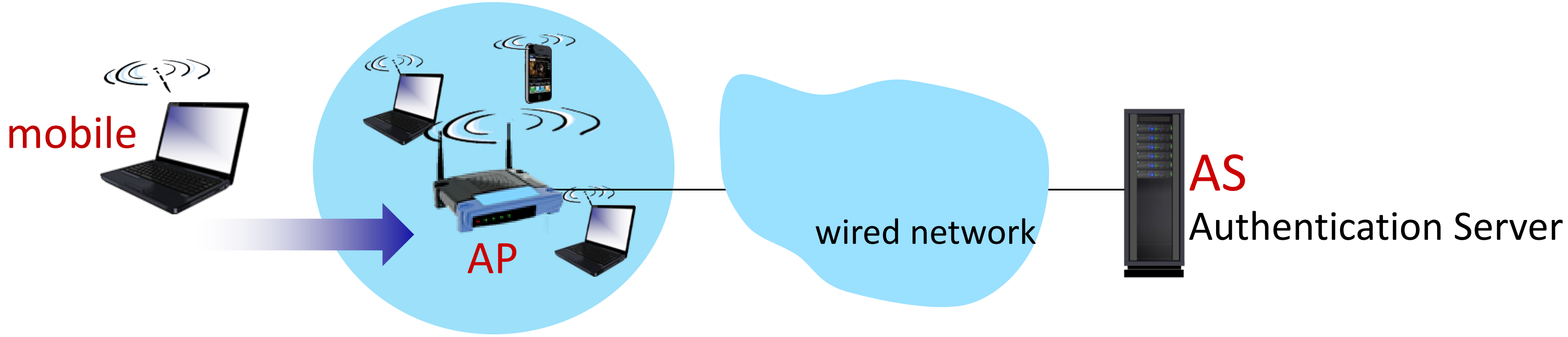
IPsec summary

- 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 8 outline

- What is network security?
- Principles of cryptography
- Authentication, digital signatures, message integrity, shared key agreement
- 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

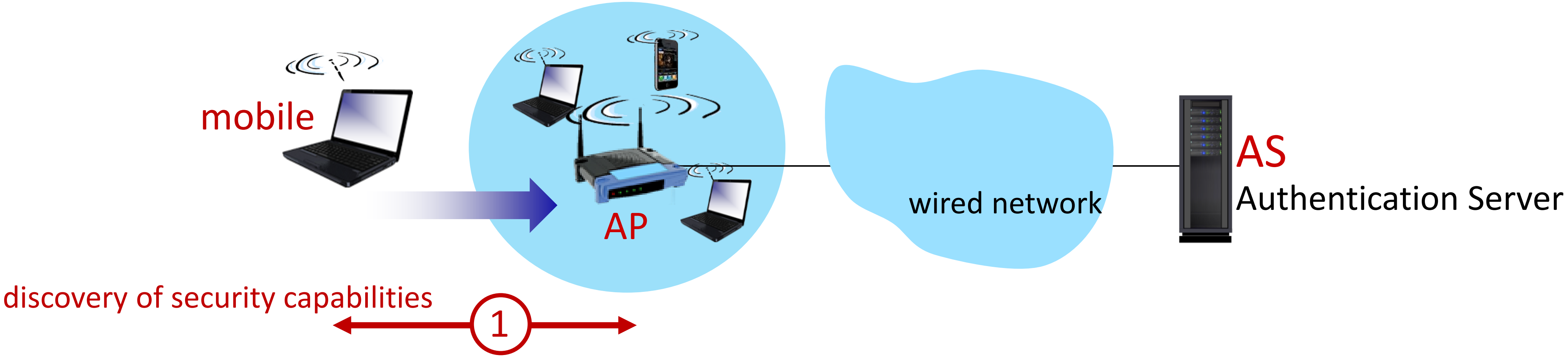
802.11: authentication, encryption



Arriving mobile (client) must:

- associate with access point: (establish) communication over wireless link
- perform mutual authentication with network
- encrypt over-the-air frames, during authentication and after (i.e., user data frames)

802.11: authentication, encryption

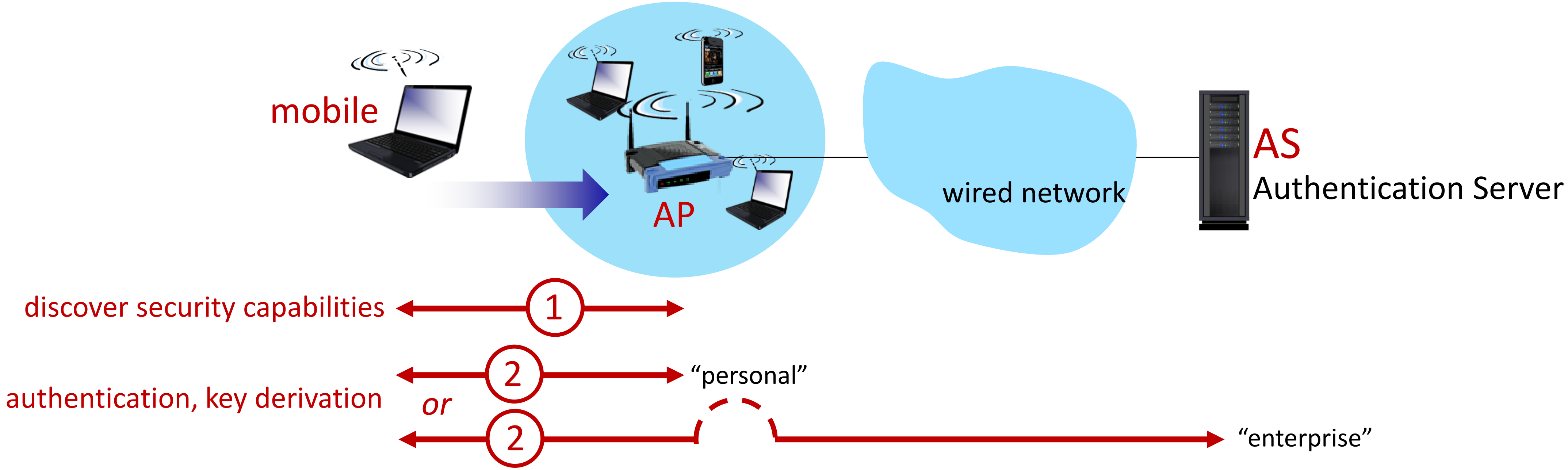


1 discovery of security capabilities:

- AP advertises its presence via `beacon` frame or device sends `probe request` frame
- AP sends it forms of authentication and encryption provided
- device requests specific forms authentication, encryption desired

initially message exchange: device not authenticated, encryption keys not known

802.11: authentication, encryption



2 mutual authentication and shared symmetric key derivation:

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

WiFi security standards: personal, enterprise

WPA (WiFi Protected Access) Security

WEP
(Wired
Equivalent Privacy)

WPA
2004

WPA2
2004

WPA3
2018

Personal:

- common in home, small offices
- *same password* for all devices
- *protocol between user device, AP*
- WPA2/3-PSK (pre-shared key)

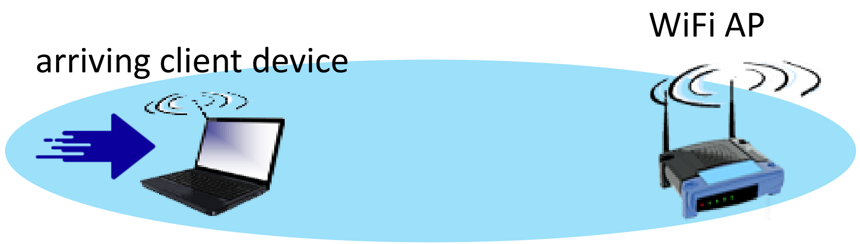
Enterprise:

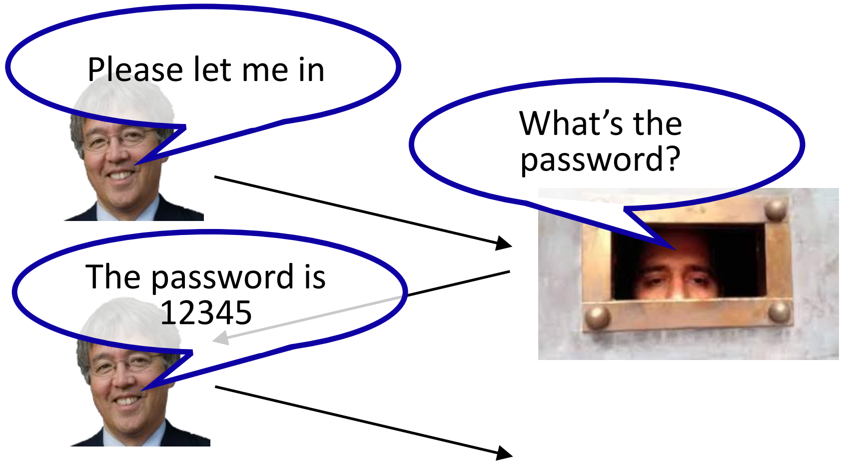
- common in enterprises
- password- and/or certificate-based
- *different per-user passwords*
- *protocols among user device, AP, authentication server* (e.g., RADIUS)
- EAP(Extensible Authentication Protocol: EAP-TLS, EAP-TTLS plus many more

WiFi security: infrastructure

Personal

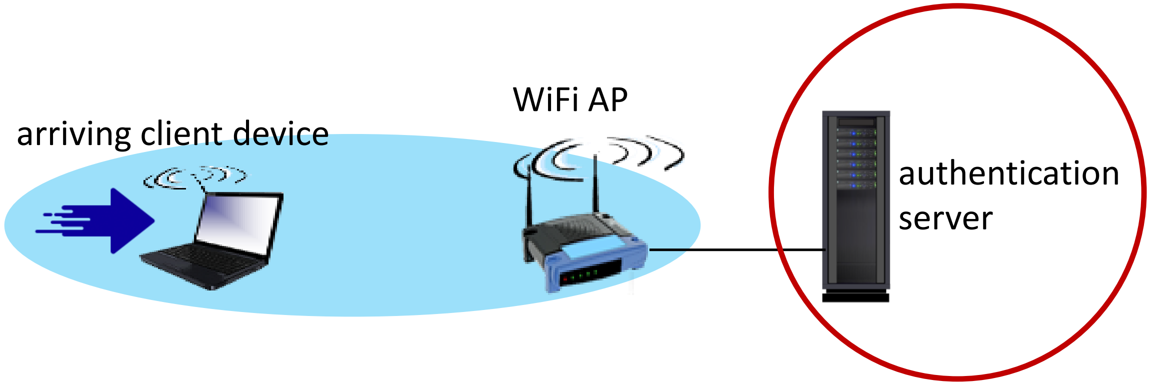
- between user device and AP
- WPA2/3-PSK: password known to device, AP

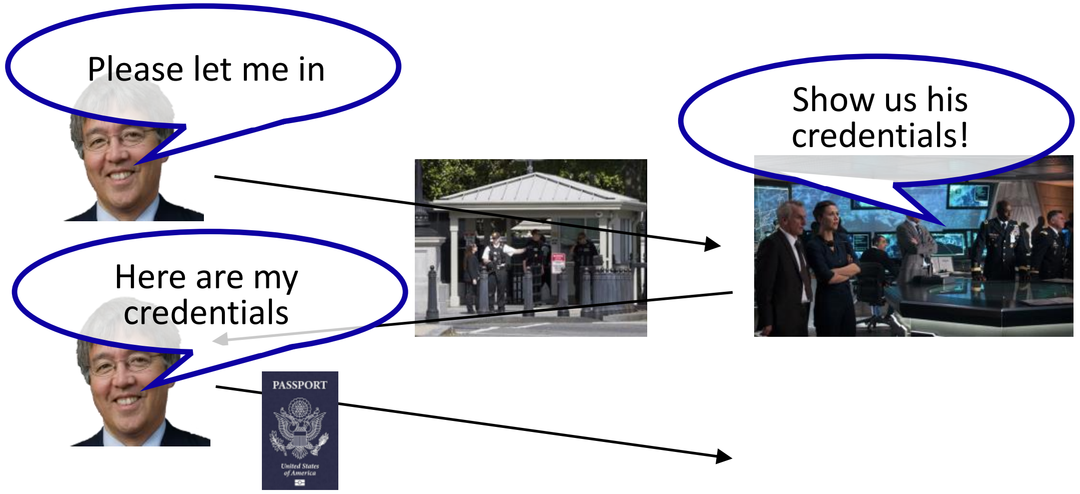




Enterprise

- mostly between user device authentication server *(AP acts as pass through)*





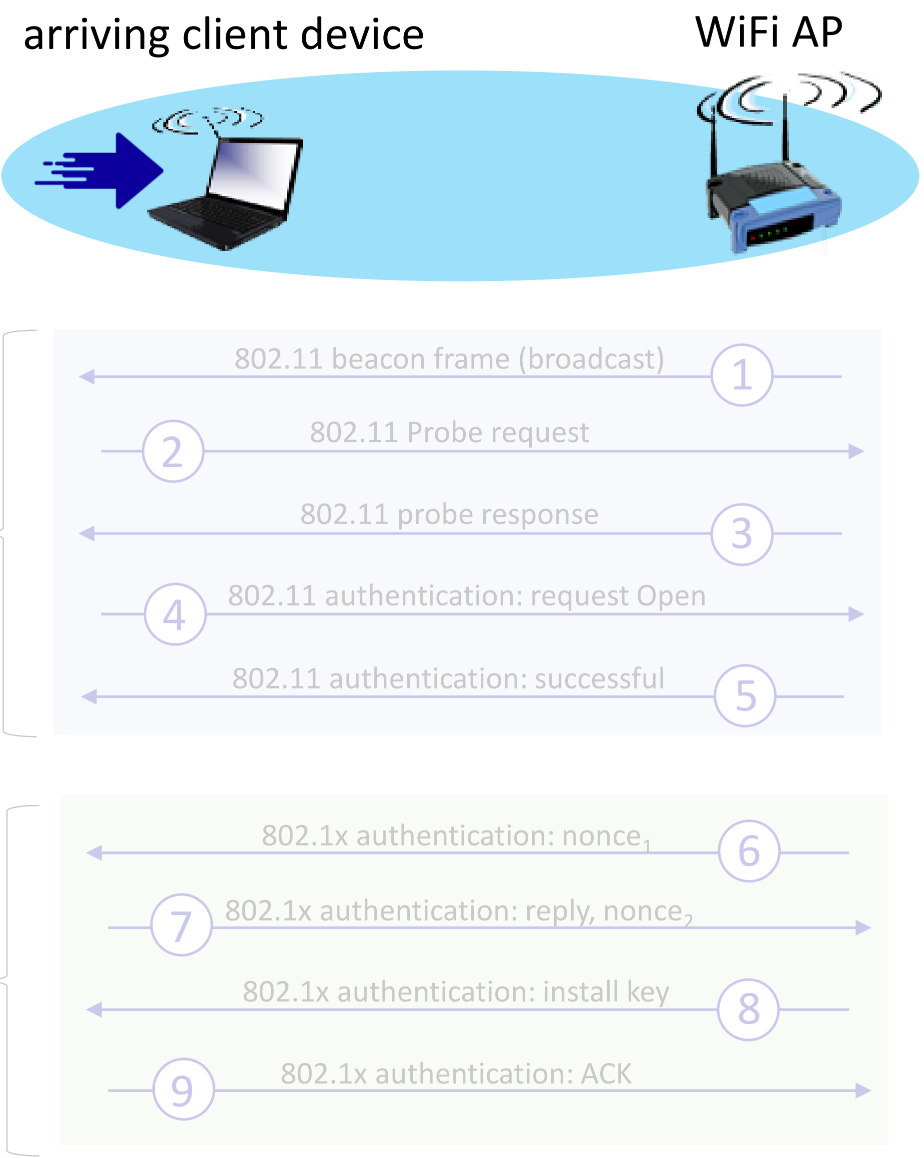
WPA2-PSK (pre-shard key)

- same password configured on AP for all devices
- password known ahead of time ("pre-shared") by devices

Two "phases" of WPA2-PSK authentication

802.11 open "authentication":
client associates with AP
(no actual "authentication" yet)

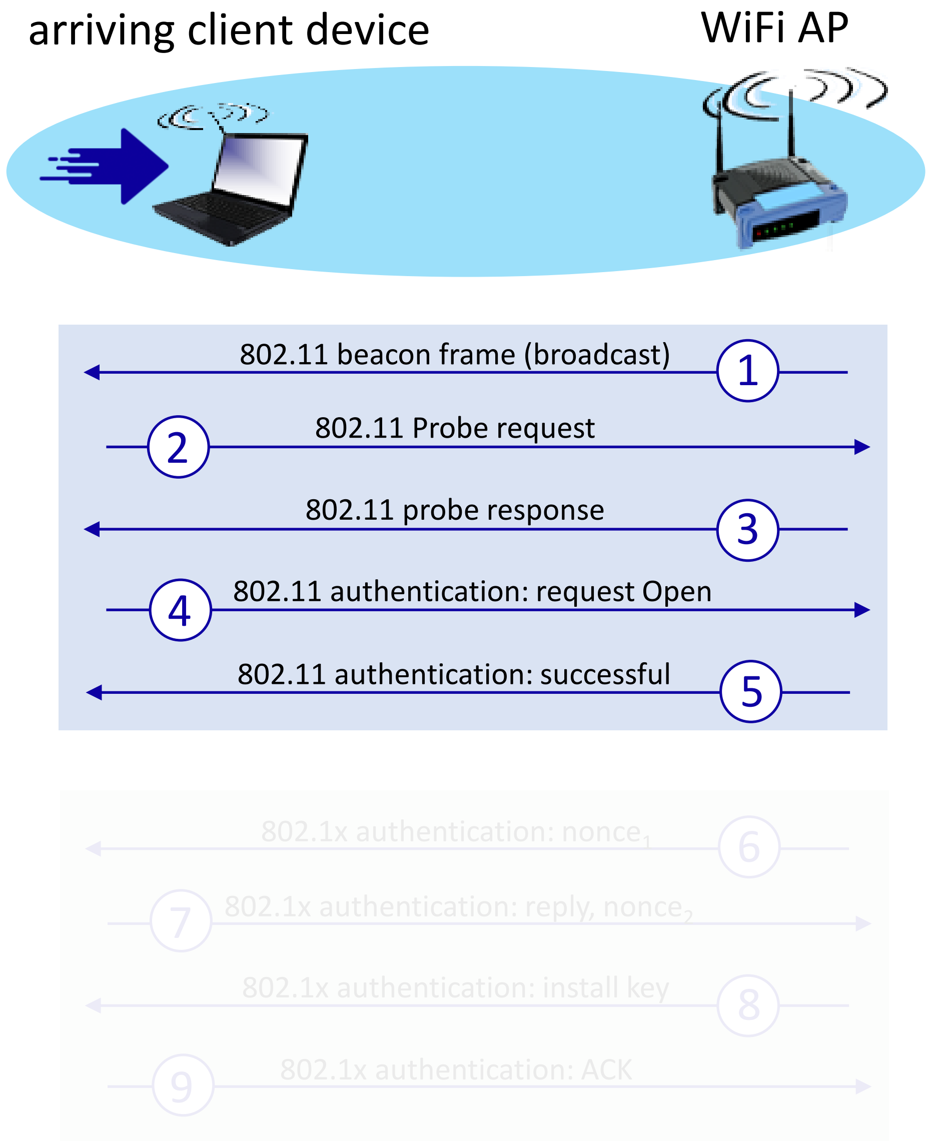
802.1x WPA2/3-PSK:
authentication, generate new
AES session encryption key



WPA2-PSK: phase 1

1. AP broadcasts 802.11 beacon frames (packet 1 in Wireshark trace*)
2. Client sends 802.11 Probe Request (packet 2)
3. AP responds with 802.11 Probe Response (packet 3)
4. Client sends 802.11 Authentication frame, requesting Open System," i.e., *no password* authentication (packet 4)
5. AP responds with 802.11 Authentication frame (packet 5), with "Status Code: Successful"

... at this point, "real" authentication has not yet happened!

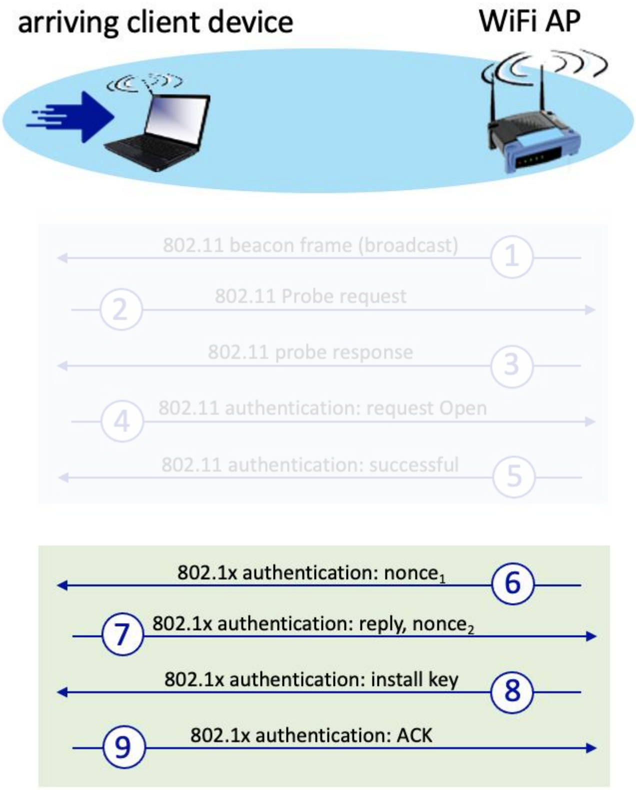


* http://gaia.cs.umass.edu/kurose/carioca_Iphone_association_selected_packets_onlyV2.pcapng

WPA2-PSK: phase 2 - 4-way EAPOL handshake

EAPOL: Extensible Authentication Protocol over LAN

- client, AP have established (unsecured) 802.11 session, know shared secret (password)
- 4-way EAPOL handshake: use shared secret, nonces to generate *new one-time session key* for encrypting this session:
  - *shared secret:* used to encrypt, and provide message integrity (MIC) for EAPOL messages
  - *MIC nonces:* guarantee liveness, protect against replay attacks
  - *new, one-time AES session key:* protects against brute force decryption attacks on shared secret



4-way EAPOL handshake for WPA2-PSK

WPA2-PSK: phase 2

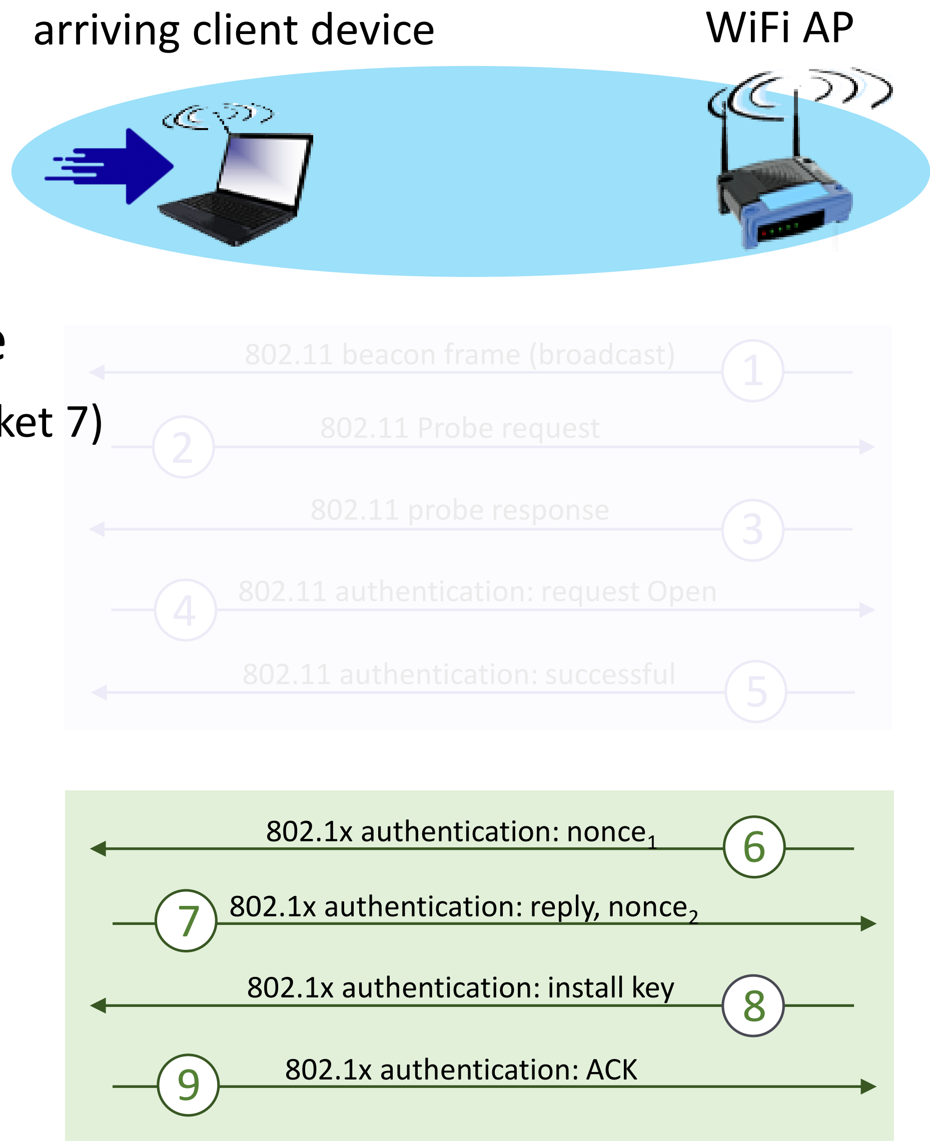
6 AP send 802.1x EAPOL message: WPA2-PSK (aka RSN), nonce$_1$ (packet 6 in Wireshark trace*)

7 Client uses shared secret and nonce$_1$ to generate reply with MIC, also generates, sends nonce$_2$ (packet 7)

8 AP checks MIC (verifying client liveness), generates new AES session key using shared secret, MIC, sends to client (packet 8)

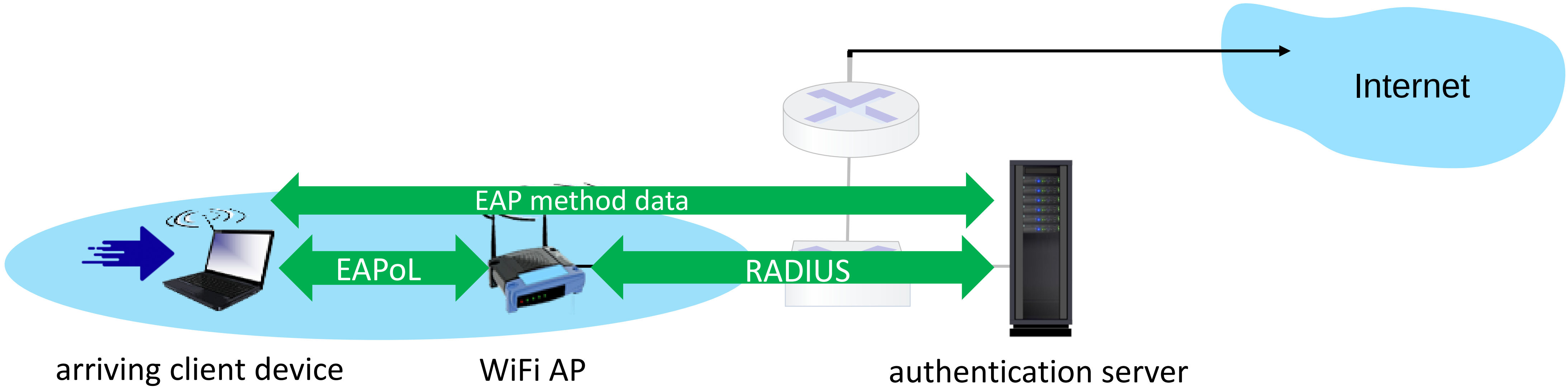
9 Client verifies MIC, sends ACKs(packet 9), installs AES session key ... *both sides ready to go!*

... at this point, AES encryption key known on both sides (note: shared secret never actually exchanged!)



* http://gaia.cs.umass.edu/kurose/carioca_Iphone_association_selected_packets_onlyV2.pcapng

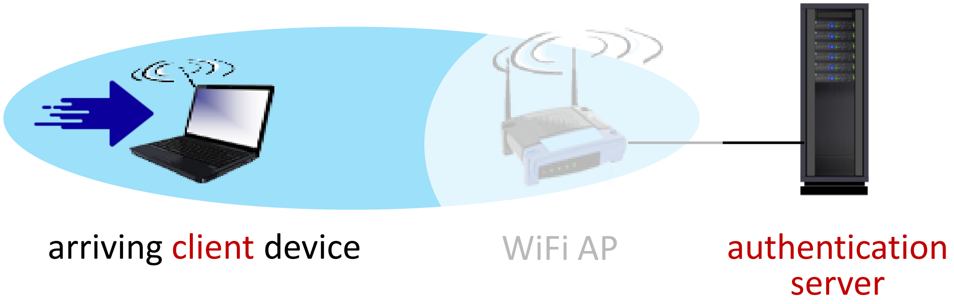
802.1x Enterprise Authentication Framework



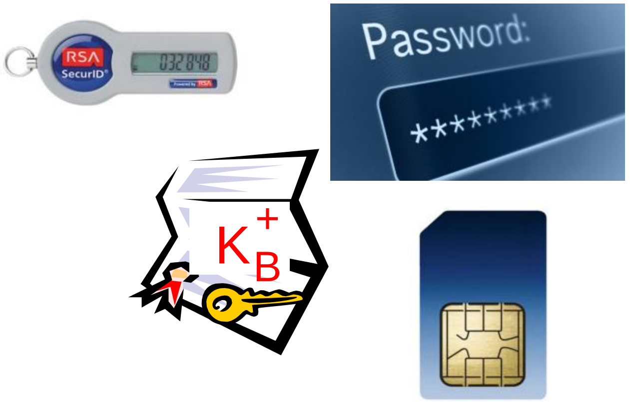
Supplicant	***Authenticator***	***Authentication server***	
Client device needing authentication before being allowed network access	relays messages between client and authentication server to authenticate client, authorize client use of WLAN	server with clients' authentication and access rights information	***802.1x enterprise terminology***

802.1x Enterprise: Extensible Authentication Protocols: EAPs

- EAP primarily between client, authentication server (e.g., RADIUS server)
  - AP acts mostly as pass through
- credentials stored at clients, authentication server

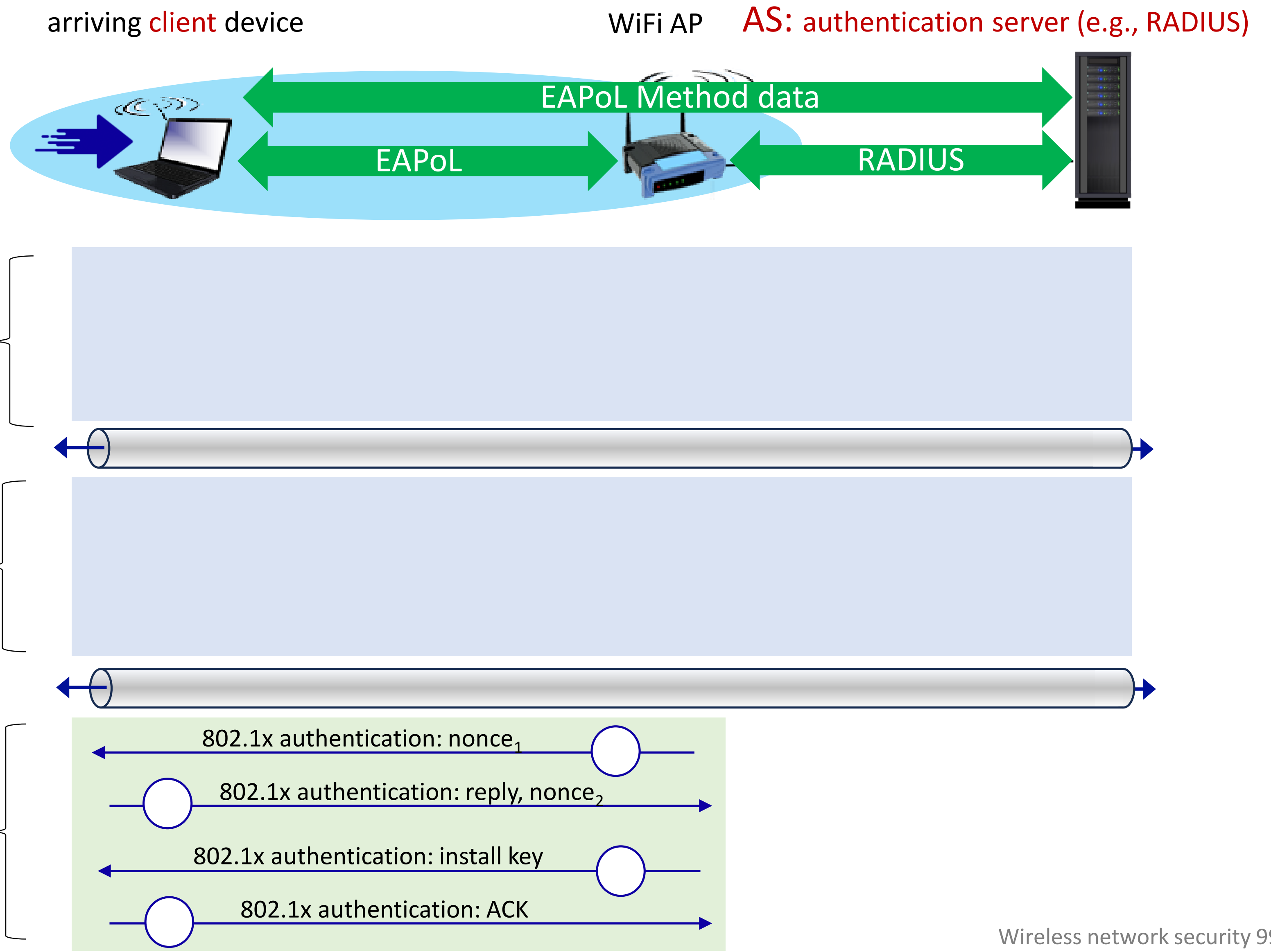


- ~40 different EAP protocols (!)
  - Differing in credentials: passwords, certificates, SIM cards, tokens ....
  - commonly used : EAP-MD-5, EAP-TLS, EAP-PEAP, EAP-TTLS, EAP-Fast, and Cisco LEAP. EAP-MD-5



EAP-TTLS: Tunneled Transport Layer Security [RFC 5281]

Two “phases” of EAP-TTLS



Phase 1: client authenticates AS (e.g., via server public key), TLS tunnel established between client, AS

Phase 2: AS authenticates client (e.g., via client password) within encrypted tunnel, tunnel torn down

4-way EAPOL handshake *to encrypt local wireless channel* (not for authentication, which is already done in phases 1,2) as in WPA2-PSK.

EAP-TTLS: message exchange phase 1 (simplified details)

3 After some handshaking, client provides name to A

4 AP: a client is seeking authentication

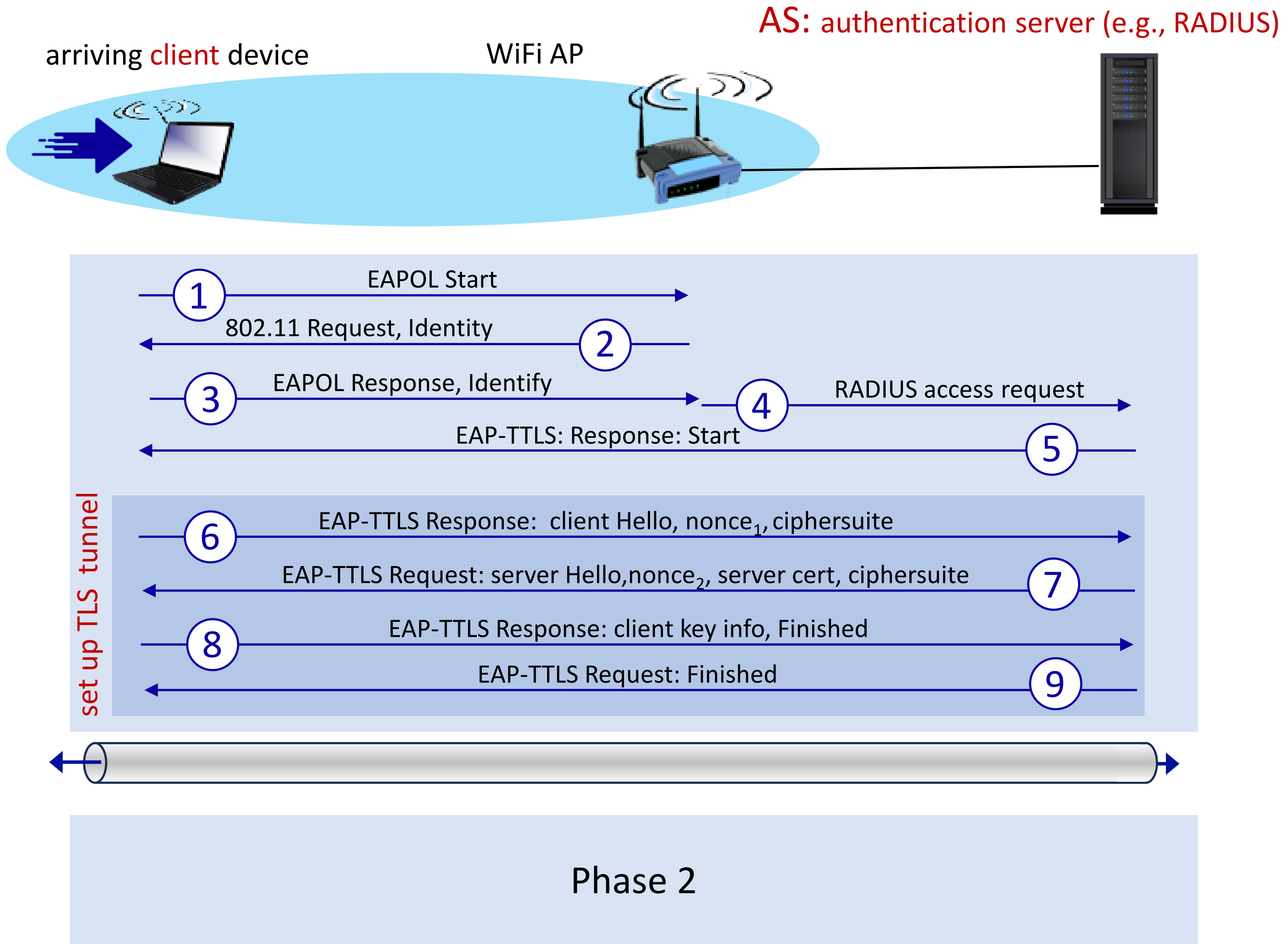
5 RADIUS server: initiates EAP-TTLS with client

6 Client sends $nonce_1$, acceptable encryption schemes to AS

7 AS response with its certificate (for authentication), $nonce_2$, chosen encryption method.

8 Client authenticates server, sends client info (encrypted using server public key)

... at this point, AS is authenticated by client, encrypted session between client, AP



EAP-TTLS: message exchange phase 2 (simplified details)

10 AS send identity request to client

11 Client responds with identity

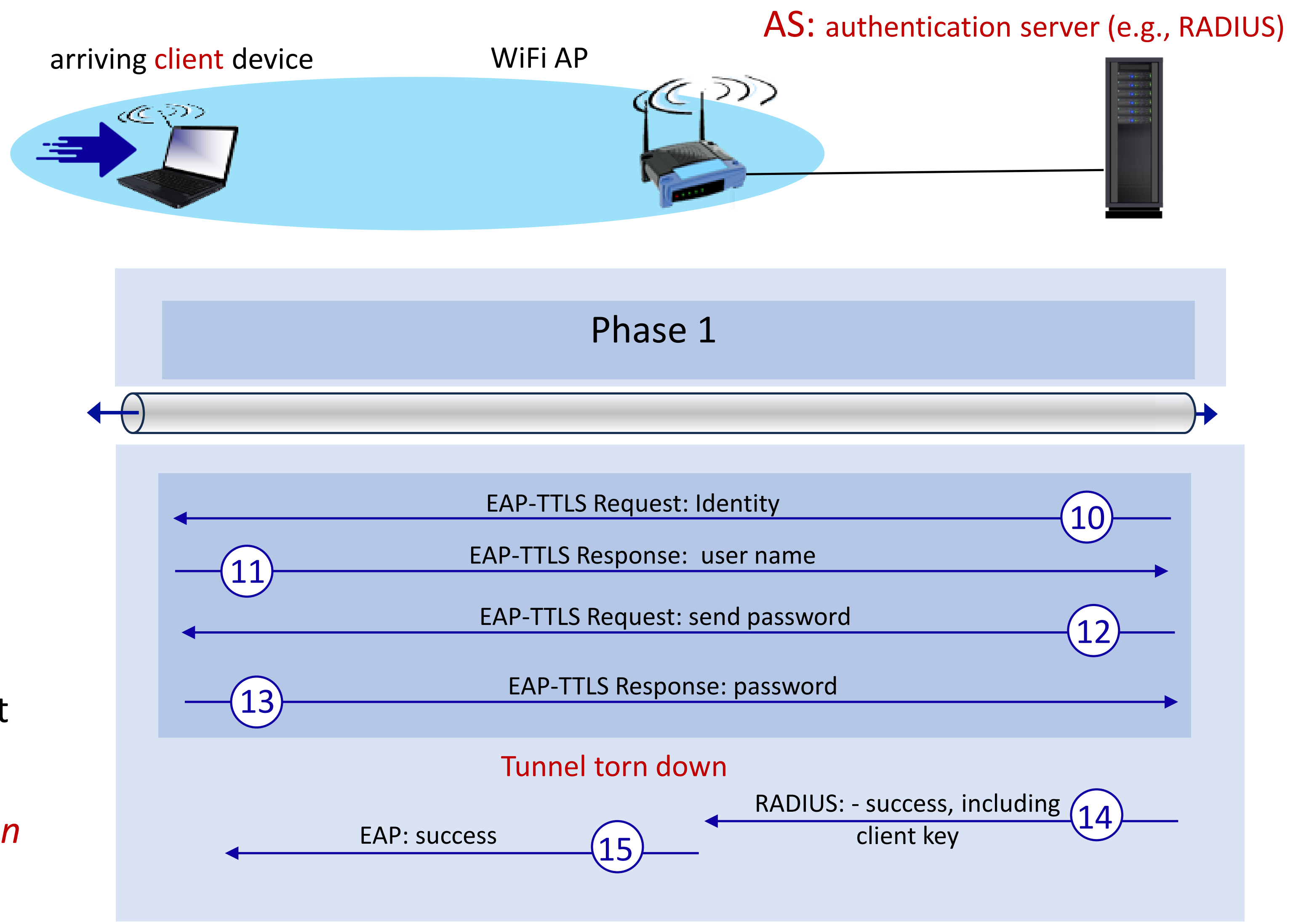
12 AS challenge - send password, or other credential, depending on EAP-TTLS authentication method

13 Client responds with credential

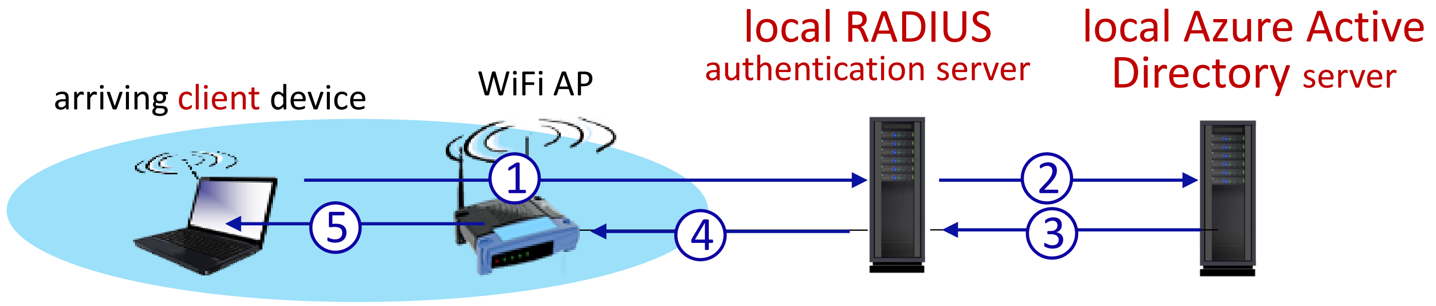
14 AS informs AP that client is authenticated, sends info needed for AP, client channel encryption

15 AP pass EAP success message to client

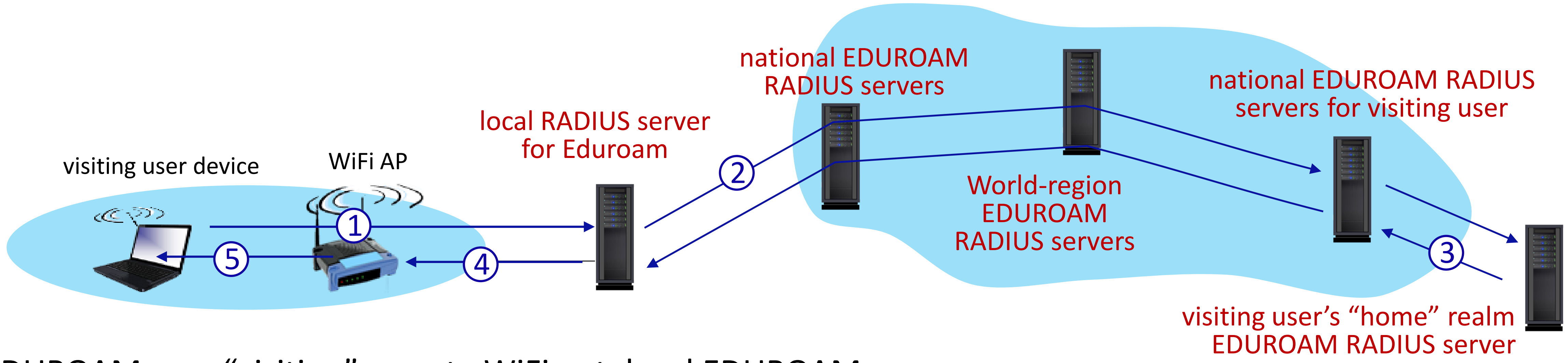
Next: client/AP agree on channel encryption using EAPoL 4-way handshake, then client can run DHCP



Local Authentication Server may use proxies

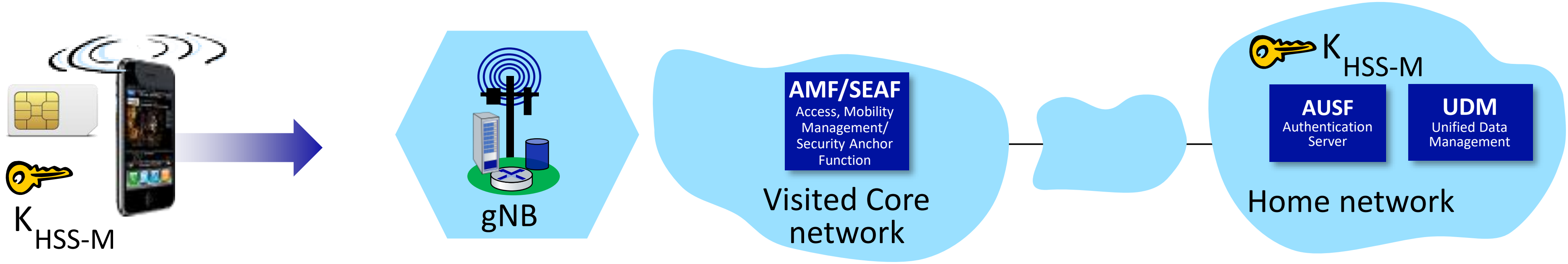


Authentication proxied from local RADIUS server to local AD server



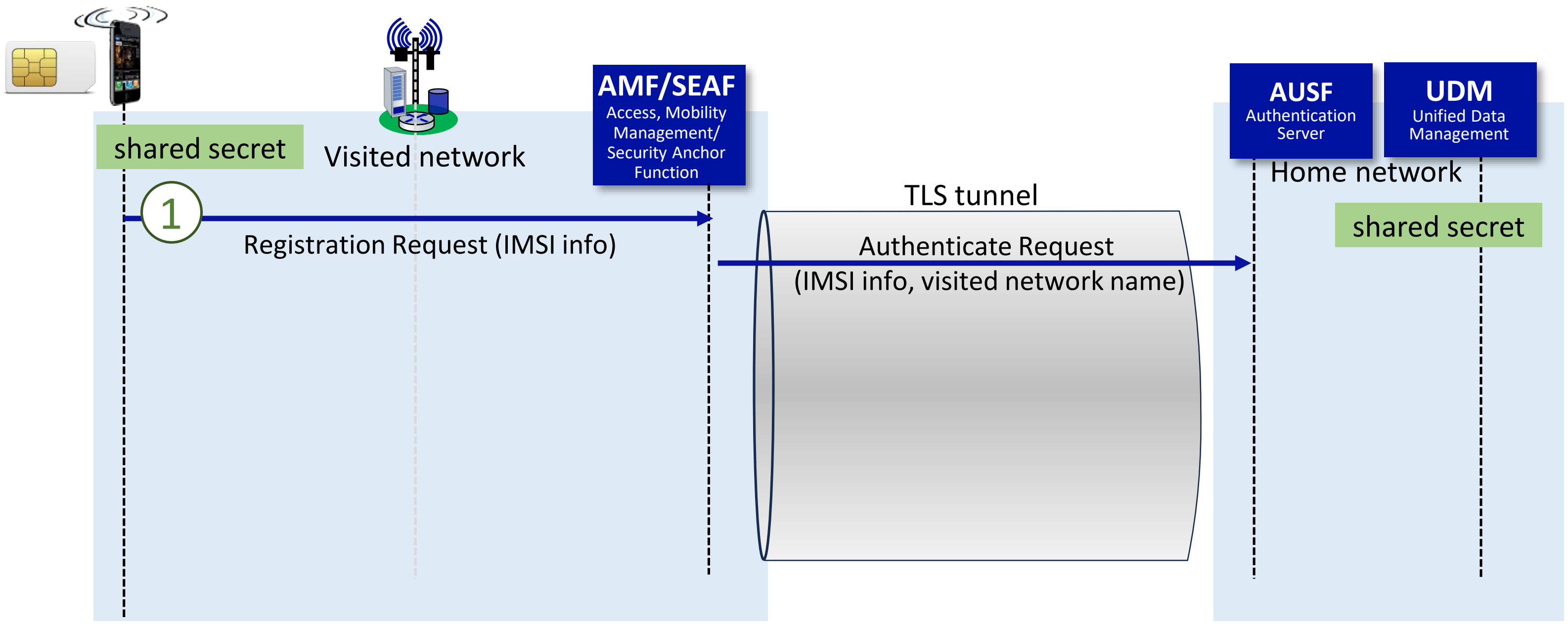
EDUROAM user "visiting" remote WiFi net: local EDUROAM RADIUS server proxies to user's home "realm" RADIUS server and accepts remote authentication decision

Authentication, encryption in 5G



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

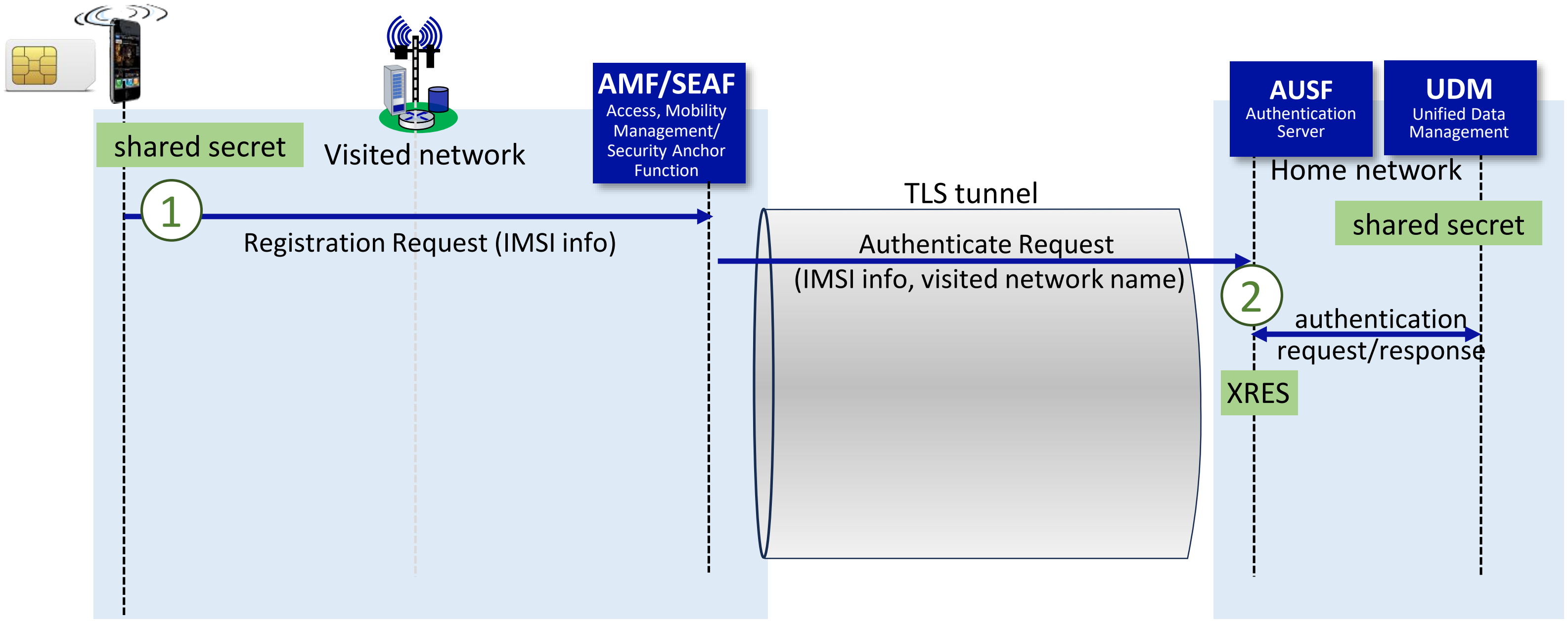
Authentication, encryption in 5G



1

- UE sends Registration Request message to visited AMF with identity (IMSI) info
- visited AMF locates AUSF in device's *home* network, establishes secure TLS tunnel to home AUSF (all messages authenticated, encrypted)

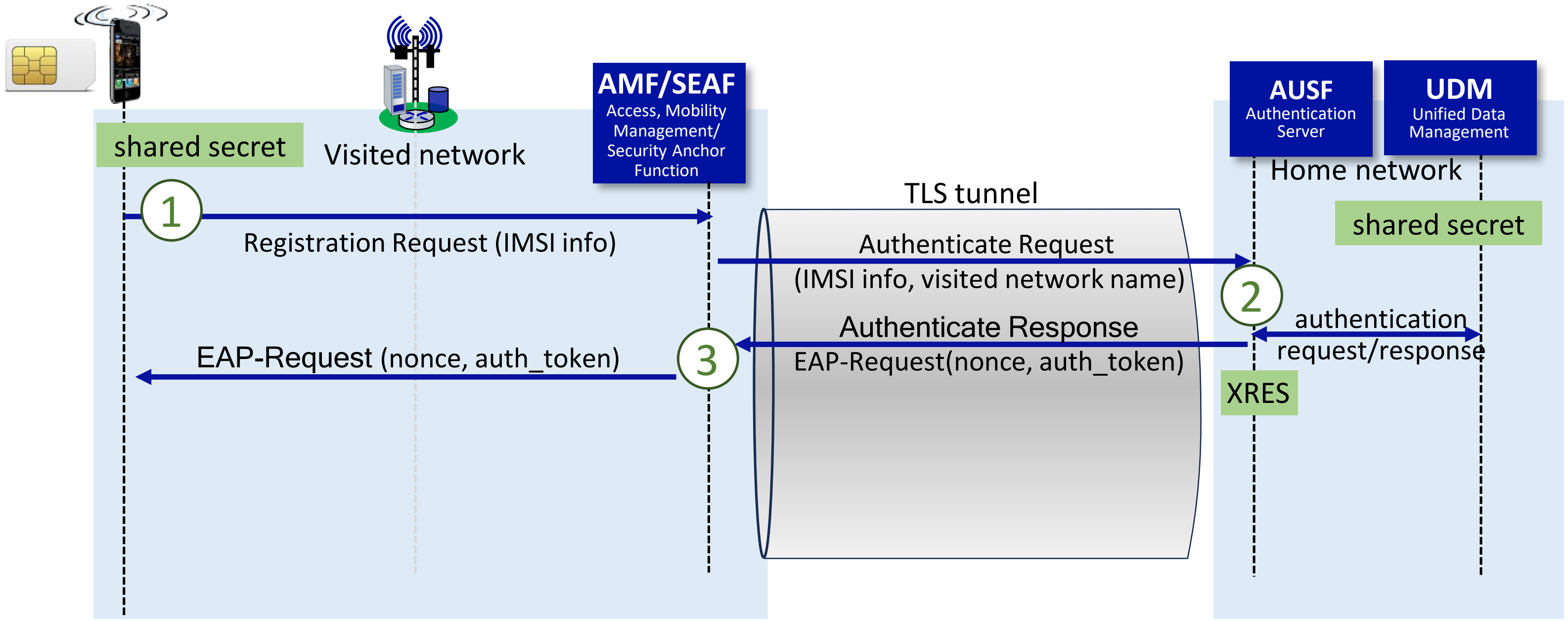
Authentication, encryption in 5G



2

- home AUSF contacts home UDM
- home UDM creates authentication token (using UE's key) and nonce, and signs (will prove to UE that home UDM generated token)
- UDM computes, saves expected response (XRES) that UE will compute

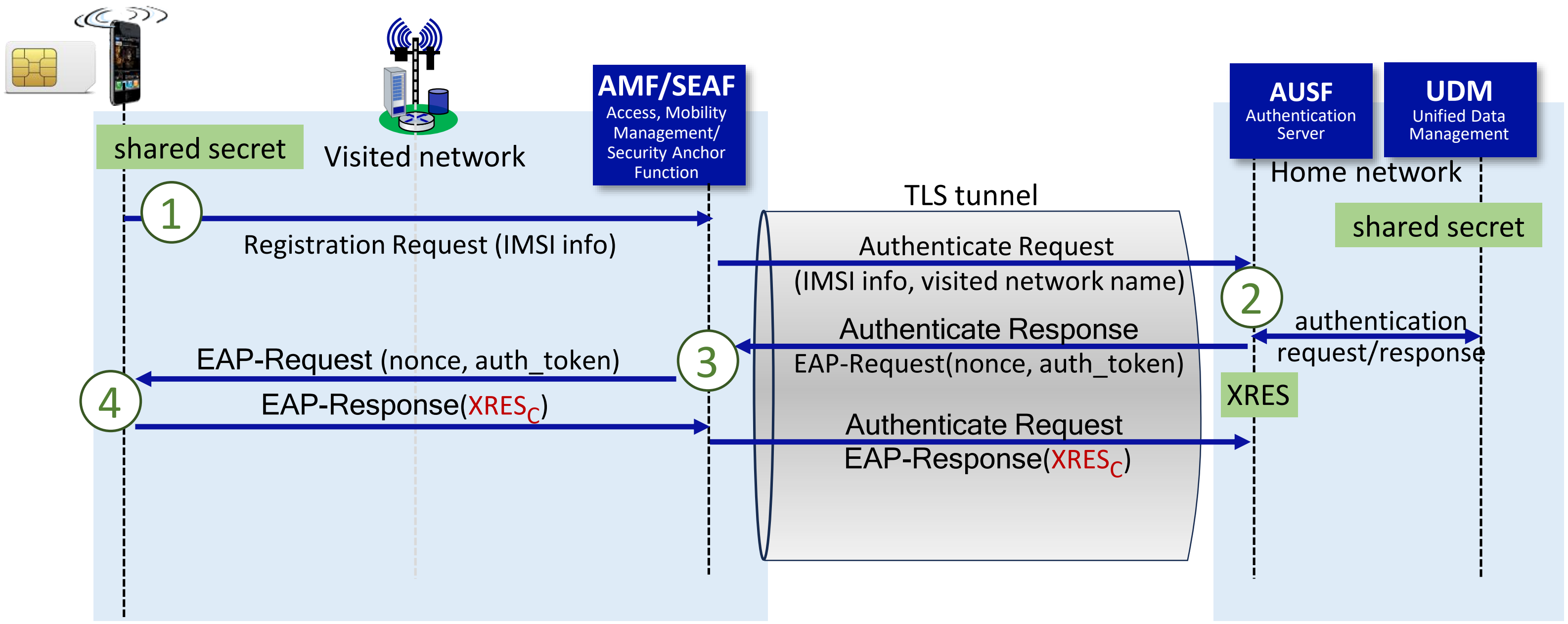
Authentication, encryption in 5G



3

- home AUSF sends authentication token, nonce to AMF, which forwards to UE

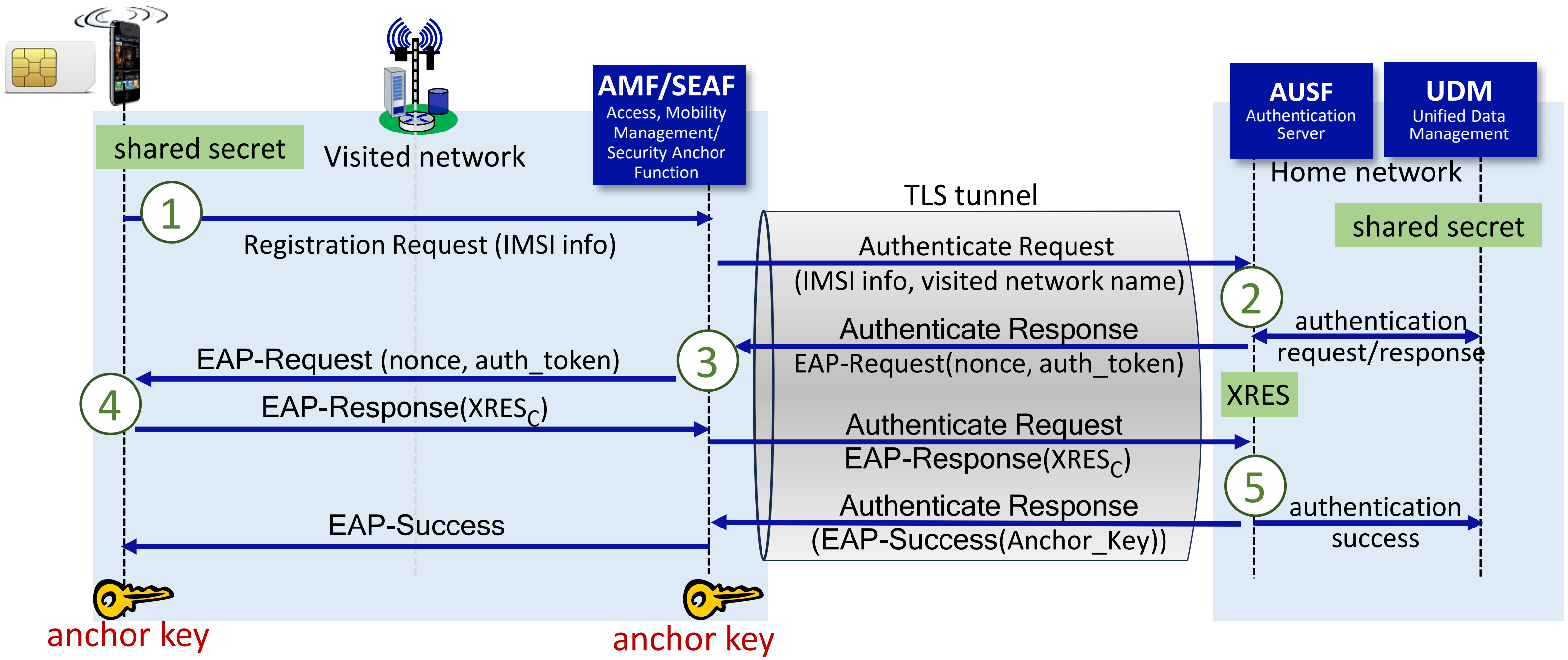
Authentication, encryption in 5G



4

- UE receives authentication token, nonce
- Given shared secret on its SIM, determines token is indeed from its home AUSF
- UE computes its expected response (XRES$_C$) sends to home AUSF

Authentication, encryption in 5G

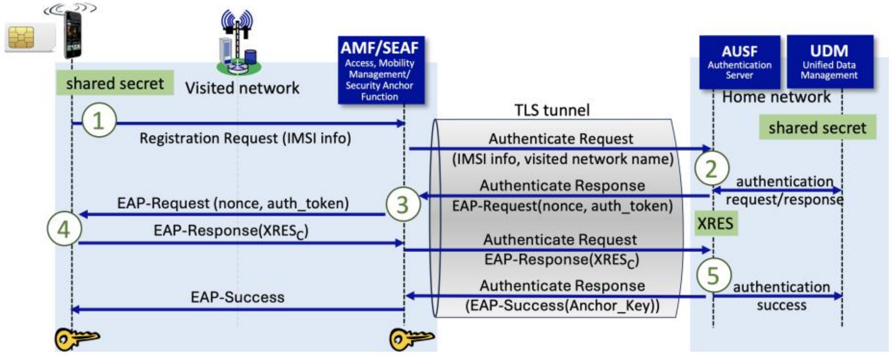


Moment of truth:

5

- AUSF computer XRES$_C$ and compares with XRES it computed earlier
- authentication success: home UDM updated
  - home AUSF generates anchor key and sends it to visited AMF (device will compute anchor key on its own
  - AMF, device use anchor key to secure message exchange in visited network

5G Authentication: observations



- *home* AUSF makes authentication comparison (used be MME in visited network in 4G)
- *home* AUSF generates anchor key to be used in visited network
  - informs UE and AMF and gNB in visited network
- note *use of crypto keys on UE's SIM* (known only to UE and home AUSF)

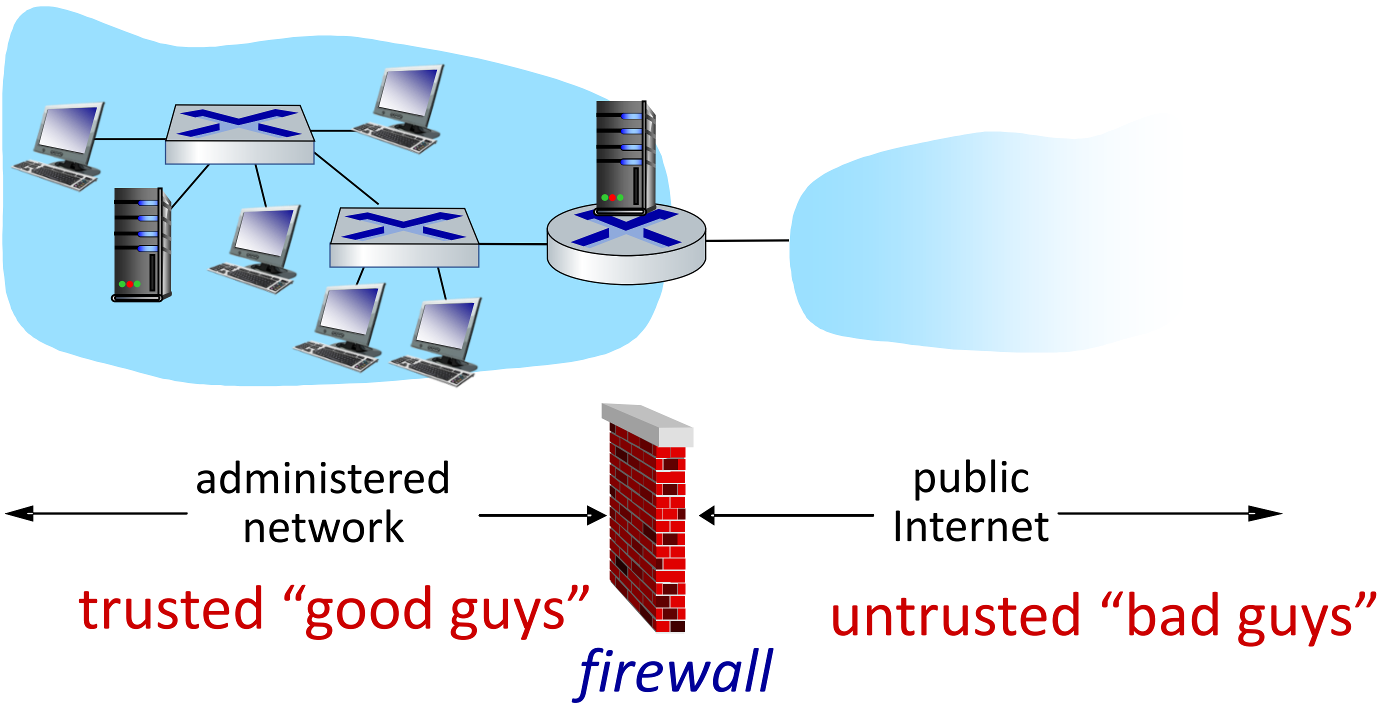
Chapter 8 outline

- 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

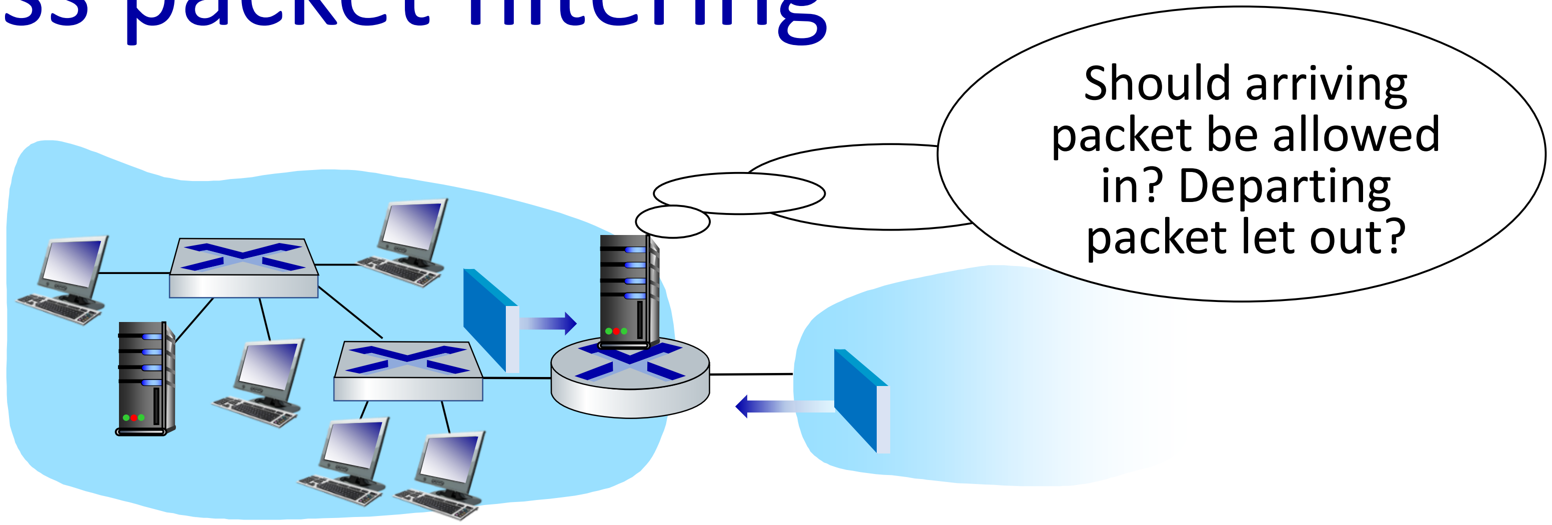
allow only authorized access to inside network

- set of authenticated users/hosts

three types of firewalls:

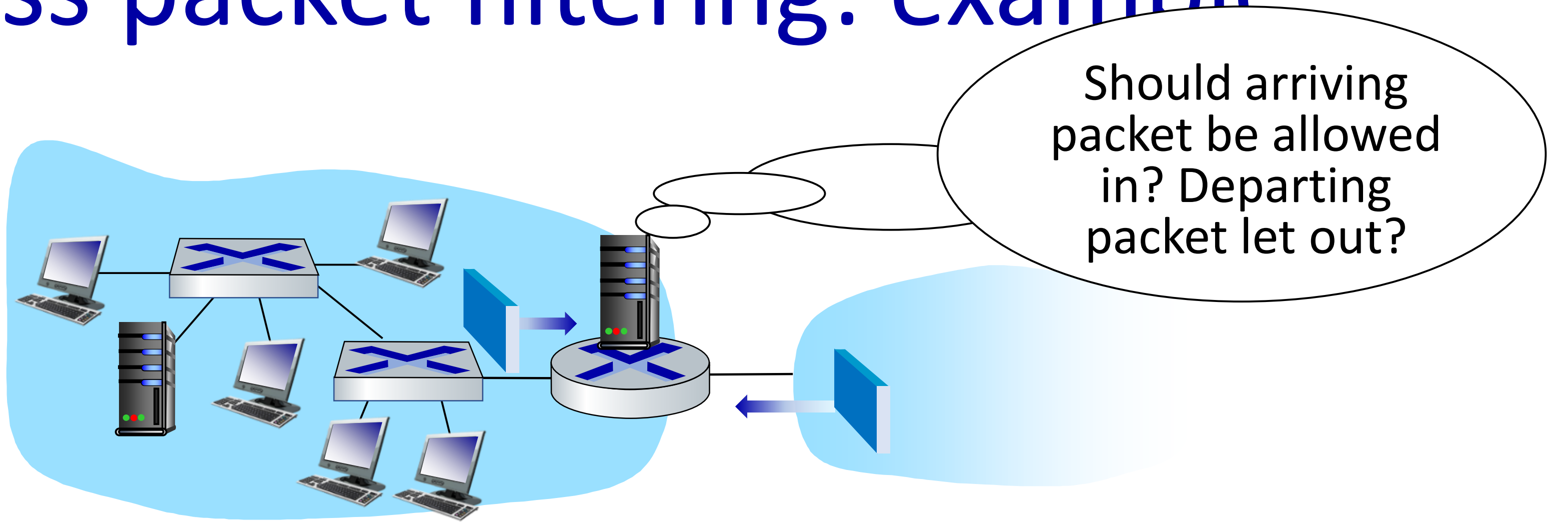
- stateless packet filters
- stateful packet filters
- application gateways

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

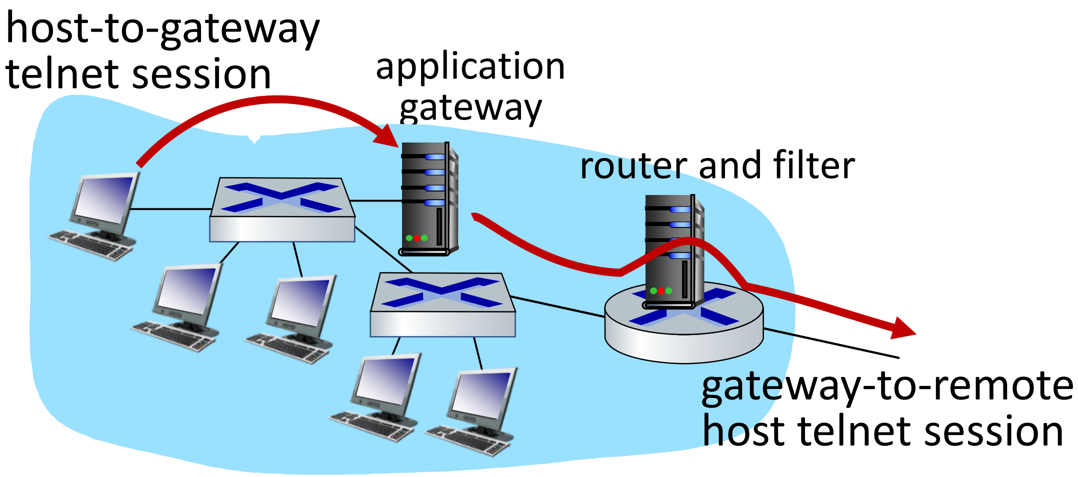
Stateful packet filtering

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

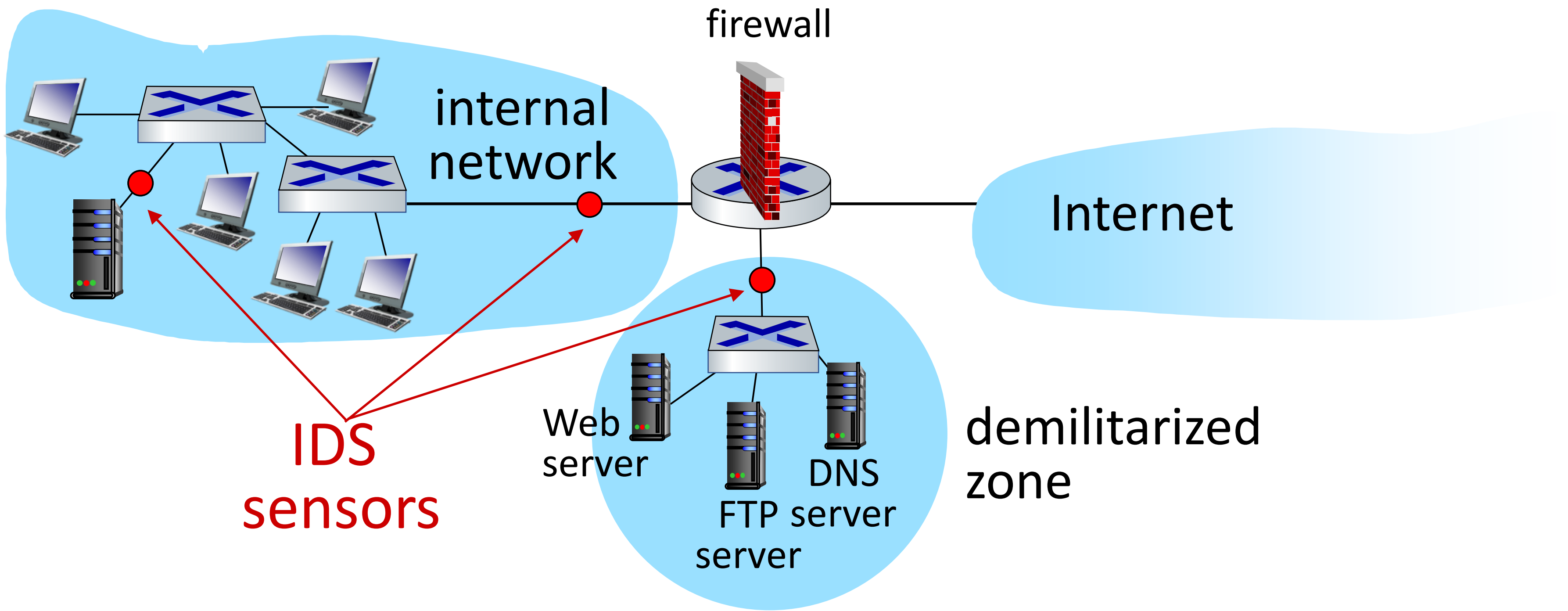
- IP spoofing: router can't know if data "really" comes from claimed source
- if multiple apps need special treatment, each has own app. gateway
- client software must know how to contact gateway
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP
- *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