last time

link-layer frame — send/receive on local network
> MAC addresses identifying source/destination machine (within local network)
>
> frame usually contains packet (potentially with info about far-fledged machines)

network-layer packet — forwarded between networks via routers
> IP(v4,v6) addresses
>
> routers have table — IP address $\rightarrow$ place to send
>
> usually contain segment/datagram

transport-layer segment/datagram
> port numbers (which program on machine)

DNS: domain name $\rightarrow$ IP addresses
> distributed, hierarchical database

quiz Q1/2

while (cannot return yet)
        cond_wait(…)

cannot use file_is_processed_cv because cannot cond wait on two CVs at once

only solution: cond-wait on single cv
        is going to require waking up sometimes when not ready
        would need changes to other functions to avoid that

quiz Q3

removing unlock()+lock() prevents othre threads from running in between

means processfile would happen with lock held

quiz Q4

correct solutions: wait for each things not done individually

cannot wait for things that are done — because could wait forever

quiz Q5

A to B [lost]: msg 1

A to B: msg 1

B to A [lost]: ack msg 1

B to A: ack msg 1

A to B: msg 2

B to A [lost]: ack msg 2

B to A: ack msg 2

…

quiz Q6

cross network time can overlap

at time 0: A to B: msg 1 (1 to send + 10 to transit)

at time 1: A to B: msg 2 (1 + 10)

at time 2: A to B: msg 3 (1 + 10)

…

at time 9: A to B: msg 10 (1 + 10)

$\rightarrow$ B receives msg 10 at time 20

names and addresses

name	address
logical identifier	location/how to locate
variable `counter`	memory address `0x7FFF9430`
DNS name `www.virginia.edu`	IPv4 address `128.143.22.36`
DNS name `mail.google.com`	IPv4 address `216.58.217.69`
DNS name `mail.google.com`	IPv6 address `2607:f8b0:4004:80b::2005`
DNS name `reiss-t3620.cs.virginia.edu`	IPv4 address `128.143.67.91`
DNS name `reiss-t3620.cs.virginia.edu`	MAC address `18:66:da:2e:7f:da`
service name `https`	port number 443
service name `ssh`	port number 22

names and addresses

name	address
logical identifier	location/how to locate
variable `counter`	memory address `0x7FFF9430`
DNS name `www.virginia.edu`	IPv4 address `128.143.22.36`
DNS name `mail.google.com`	IPv4 address `216.58.217.69`
DNS name `mail.google.com`	IPv6 address `2607:f8b0:4004:80b::2005`
DNS name `reiss-t3620.cs.virginia.edu`	IPv4 address `128.143.67.91`
DNS name `reiss-t3620.cs.virginia.edu`	MAC address `18:66:da:2e:7f:da`
service name `https`	port number `443`
service name `ssh`	port number `22`

DNS: distributed database

my
machine

ISP's
DNS server

address sent to my machine
when it connected to network
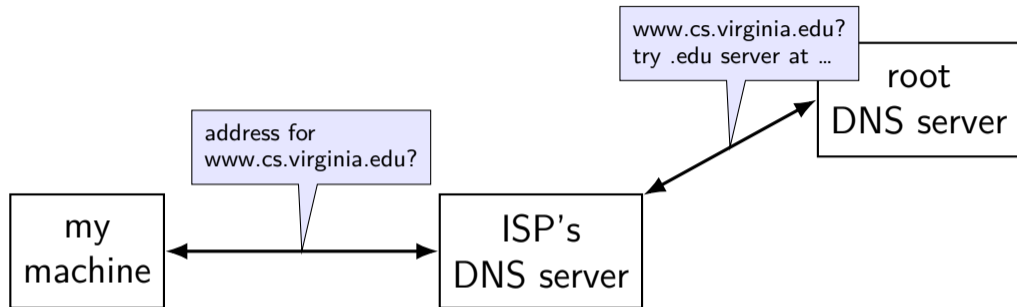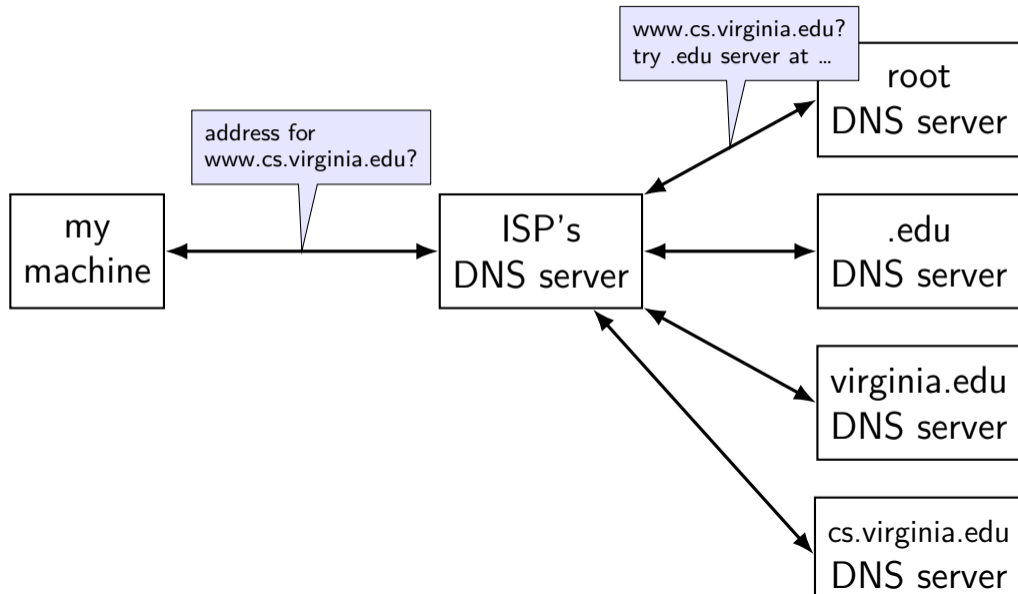
DNS: distributed database

DNS: distributed database

DNS: distributed database

DNS: distributed database

address for
www.cs.virginia.edu?

www.cs.virginia.edu?
try .edu server at …

my
machine

ISP's
DNS server

root
DNS server

.edu
DNS server

www.cs.virginia.edu =
128.143.67.11

.edu server doesn't change much
optimization: *cache* its address

check for updated version once in a while

DNS server

10

querying the root

```
$ dig +trace +all www.cs.virginia.edu
...
edu.                    172800      IN      NS      b.edu-servers.net.
edu.                    172800      IN      NS      f.edu-servers.net.
edu.                    172800      IN      NS      i.edu-servers.net.
edu.                    172800      IN      NS      a.edu-servers.net.
...
b.edu-servers.net.      172800      IN      A       191.33.14.30
b.edu-servers.net.      172800      IN      AAAA    2001:503:231d::2:30
f.edu-servers.net.      172800      IN      A       192.35.51.30
f.edu-servers.net.      172800      IN      AAAA    2001:503:d414::30
...
;; Received 843 bytes from 198.97.190.53#53(h.root-servers.net) in 8 ms
...
```

querying the edu

```
$ dig +trace +all www.cs.virginia.edu
...
virginia.edu.                    172800      IN      NS      nom.virginia.edu.
virginia.edu.                    172800      IN      NS      uvaarpa.virginia.edu.
virginia.edu.                    172800      IN      NS      eip-01-aws.net.virginia.edu.
nom.virginia.edu.       172800      IN      A       128.143.107.101
uvaarpa.virginia.edu.       172800      IN      A       128.143.107.117
eip-01-aws.net.virginia.edu. 172800 IN      A       44.234.207.10
;; Received 165 bytes from 192.26.92.30#53(c.edu-servers.net) in 40 ms
...
```

12

querying virginia.edu+cs.virginia.edu

```
$ dig +trace +all www.cs.virginia.edu
...
cs.virginia.edu.         3600         IN         NS         coresrv01.cs.virginia.edu.
coresrv01.cs.virginia.edu. 3600       IN         A          128.143.67.11
;; Received 116 bytes from 44.234.207.10#53(eip-01-aws.net.virginia.edu) in 72 ms

www.cs.Virginia.EDU.         172800         IN         A         128.143.67.11
cs.Virginia.EDU.         172800         IN         NS         coresrv01.cs.Virginia.EDU.
coresrv01.cs.Virginia.EDU. 172800 IN         A         128.143.67.11
;; Received 151 bytes from 128.143.67.11#53(coresrv01.cs.virginia.edu) in 4 ms
```

querying typical ISP's resolver

```
$ dig www.cs.virginia.edu
...
;; ANSWER SECTION:
www.cs.Virginia.EDU.          7183          IN          A          128.143.67.11
..
```

cached response

valid for 7183 more seconds

after that everyone needs to check again

exercise

suppose initially
    *.foo.com DNS server ('nameserver') = 10.2.3.4, valid 200 s
    www.foo.com = 10.1.2.3, valid 100 s

if at time 0 seconds, changed to:
    *.foo.com DNS server = 10.3.4.5, valid 100 s
    www.foo.com DNS server = 10.3.5.1, valid 400 s

ex 0: when will new DNS server/www.foo.com start being used?

ex 1: when can we shut down old DNS server?

ex 2: when can we shut down old www.foo.com?

names and addresses

name	address
logical identifier	location/how to locate
variable counter	memory address 0x7FFF9430
DNS name www.virginia.edu	IPv4 address 128.143.22.36
DNS name mail.google.com	IPv4 address 216.58.217.69
DNS name mail.google.com	IPv6 address 2607:f8b0:4004:80b::2005
DNS name reiss-t3620.cs.virginia.edu	IPv4 address 128.143.67.91
DNS name reiss-t3620.cs.virginia.edu	MAC address 18:66:da:2e:7f:da
service name https	port number 443
service name ssh	port number 22

two types of addresses?

MAC addreses: on link layer

IP addresses: on network layer

how do we know which MAC address to use?

a table on my desktop

my desktop:

```
$ arp -an
? (128.143.67.140) at 3c:e1:a1:18:bd:5f [ether] on enp0s31f6
? (128.143.67.236) at <incomplete> on enp0s31f6
? (128.143.67.11) at 30:e1:71:5f:39:10 [ether] on enp0s31f6
? (128.143.67.92) at <incomplete> on enp0s31f6
? (128.143.67.5) at d4:be:d9:b0:99:d1 [ether] on enp0s31f6

…
```

network address to link-layer address + interface

only tracks things directly connected to my local network
    non-local traffic sent to local router

how is that table made?

ask all machines on local network (same switch)

"Who has 128.148.67.140"

the correct one replies

addendum re: routing

routers track network + next router for IP address range

IPv4 addresses and routing tables
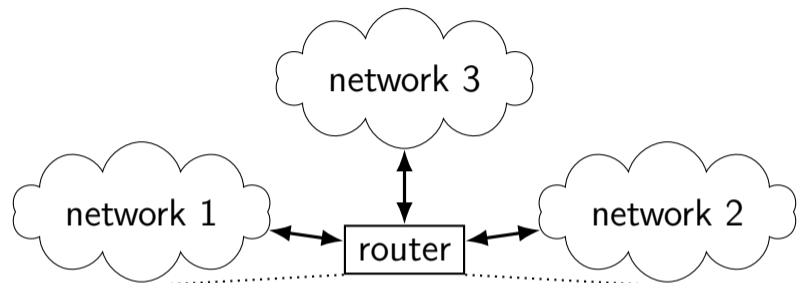
if I receive data for...	send it to...
128.143.0.0—128.143.255.255	network 1, 11.4.3.2
192.107.102.0–192.107.102.255	network 1, 11.4.3.2
...	...
4.0.0.0–7.255.255.255	network 2, 12.4.6.4
64.8.0.0–64.15.255.255	network 2, 45.4.0.1
...	...
anything else	network 3, 199.44.33.1

URL / URIs

Uniform Resource Locators (URL)
   tells how to find "resource" on network
   uniform — one syntax for multiple protocols (types of servers, etc.)

Unifrom Resources Identifiers
   superset of URLs

URI examples

```
https://kytos02.cs.virginia.edu:443/cs3130-spring2023/
                 quizzes/quiz.php?qid=02#q2

https://kytos02.cs.virginia.edu/cs3130-spring2023/
                 quizzes/quiz.php?qid=02

https://www.cs.virginia.edu/

sftp://cr4bd@portal.cs.virginia.edu/u/cr4bd/file.txt

tel:+1-434-982-2200

//www.cs.virginia.edu/~cr4bd/3130/S2023/
/~cr4bd/3130/S2023
```
scheme and/or host implied from context

URI generally

```
scheme://authority/path?query#fragment
```
scheme: — what protocol

//authority/
    authoirty = user@host:port OR host:port OR user@host OR host

path
    which resource

?query — usually key/value pairs

#fragment — place in resource

most components (sometimes) optional

autoconfiguration

problem: how does my machine get IP address

otherwise:
      have sysadmin type one in?
      just choose one?
      ask machine on local network to assign it

autoconfiguration

problem: how does my machine get IP address

otherwise:
      have sysadmin type one in?
      just choose one?
      ask machine on local network to assign it

autoconfiguration

problem: how does my machine get IP address

otherwise:
> have sysadmin type one in?
> just choose one?
> ask machine on local network to assign it

often local router machine runs service to assign IP addresses
> knows what IP addresses are available
> sysadmin might configure in mapping from MAC addresses to IP addresses

DHCP high-level

protocol done over UDP

but since we don't have IP address yet, use `0.0.0.0`

and since we don't know server address, use `255.255.255.255`
= "everyone on the local network"

local server replies to request with address + time limit

later: can send messages to local server to renew/give up address

DHCP high-level

protocol done over UDP

but since we don't have IP address yet, use `0.0.0.0`

and since we don't know server address, use `255.255.255.255`
 = "everyone on the local network"

local server replies to request with address + time limit

later: can send messages to local server to renew/give up address

exercise: why time limit?

DHCP "lease"

rather than getting address forever

but DHCP has way of releasing taken address

why impose a time limit

network address translation

IPv4 addresses are kinda scarce

solution: *convert* many private addrs. to one public addr.

locally: use private IP addresses for machines

outside: private IP addresses become a single public one

commonly how home networks work (and some ISPs)

implementing NAT

remote host + port	outside local port number	inside IP	inside port number
128.148.17.3:443	54033	192.168.1.5	43222
11.7.17.3:443	53037	192.168.1.5	33212
128.148.31.2:22	54032	192.168.1.37	43010
128.148.17.3:443	63039	192.168.1.37	32132

table of the translations

need to update as new connections made

upcoming lab

request + receive message split into pieces

you are responsible for:
    requesting parts in order
    resending requests if messages lost/corrupted

"acknowledge" receiving part X to request part X+1

upcoming lab

request + receive message split into pieces

you are responsible for:
    requesting parts in order
    resending requests if messages lost/corrupted

"acknowledge" receiving part X to request part X+1

protocol

GETx — retrieve message x (x = 0, 1, 2, or 3)

    other end acknowledges by giving data

    if they don't reply, you need to send again

    higher numbered messages have errors/etc. that are harder to handle

ACKn

    request message $n + 1$ by acknowledging message n

    not quite same purpose as acknowledgments in prior examples

    (in lab, the response is your 'acknowledgment' of your request;

    you retry if you don't get it)

protocol

GETx — retrieve message x (x = 0, 1, 2, or 3)
> other end acknowledges by giving data
> if they don't reply, you need to send again
> higher numbered messages have errors/etc. that are harder to handle

ACKn
> request message $n + 1$ by acknowledging message n
> not quite same purpose as acknowledgments in prior examples
> (in lab, the response is your 'acknowledgment' of your request;
> you retry if you don't get it)

callback-based programming (1)

```
/* library code you don't write */
/* in the lab: part of waitForAllTimeoutsAndMessagesThenExit()
void mainLoop() {
    while (notExiting) {
        Event event = waitForAndGetNextEvent();
        if (event.type == RECIEVED) {
            recvd(...);
        } else if (event.type == TIMEOUT) {
            (event.timeout_function)(...);
        }
        ...
    }
}
```

callback-based programming (2)

```
/* your code, called by library */
void recvd(...) {
    ...
    setTimeout(..., timerCallback, ...);
}

void timerCallback(...) {
    ...
}

int main() {
    send(.../* first message */);
    ... /* other initial setup */
    waitForAllTimeoutsAndMessagesThenExit(); // runs mainLoop
}
```

callback-based programming

writing scripts in a webpage

many graphical user interface libraries

sometimes servers that handle lots of connections

firewalls

don't want to expose network service to everyone?

solutions:
    service picky about who it accepts connections from
    filters in OS on machine with services
    filters on router

later two called "firewalls"

firewall rules examples?

ALLOW tcp port 443 (https) FROM everyone

ALLOW tcp port 22 (ssh) FROM my desktop's IP address

BLOCK tcp port 22 (ssh) FROM everyone else

ALLOW from address X to address Y

…

secure communication context

"secure" communication

mostly talk about on network

between *principals* ≈ people/servers/programs

but same ideas apply to, e.g., messages on disk
     communicating with yourself

A to B

running example: A talking with B
> maybe sometimes also with C

attacker E — eavesdropper
> passive
> gets to read all messages over network

attacker M — machine-in-the-middle
> active
> gets to read and replace and add messages on the network

privileged network position

intercept radio signal?

control local wifi router?
> may doesn't just forward messages

compromise network equipment?

send packets with 'wrong' source address
> called "spoofing"

fool DNS servers to 'steal 'name?

fool routers to send you other's data?

possible security properties? (1)

what we'll talk about:

confidentiality — information shared only with those who should have it

authenticity — message genuinely comes from right principal (and not manipulated)

possible security properties? (2)

important ones we won't talk about…:

repudiation — if A sends message to B, B can't prove to C it came from A

    (takes extra effort to get along with authenticity)

forward-secrecy — if A compromised now, E can't use that to decode past conversations with B

anonymity — A can talk to B without B knowing who it is

…

secrets

if A is talking to B are communicating,
what stops M (machine-in-the-middle) from pretending to be B?

assumption: B knows some secret information that M does not

secrets

if A is talking to B are communicating,
what stops M (machine-in-the-middle) from pretending to be B?

assumption: B knows some secret information that M does not

start: assume A and B have a *shared secret* they both know
(and attackers do not)

(later: easier to setup assumptions)

bad ways to use shared secret

A → B: What's the password?

B → A: It's 'AbcxyMe'.

A → B: That's right! Here's my confidential information.

bad ways to use shared secret

A → B: What's the password?

B → A: It's 'Abc\$xyM\$e'.

A → B: That's right! Here's my confidential information.

well, this doesn't really help:

> against E (eavesdropper), who can read the password AND confidential info
>
> against M (machine-in-the-middle), who can also pretend to be A for B

symmetric encryption

some magic math!

we'll be given two functions by expert:
    encrypt: $E(\text{key}, \text{message}) = \text{ciphertext}$
    decrypt: $D(\text{key}, \text{ciphertext}) = \text{message}$

key = shared secret
    ideally small (easy to share) and chosen at random
    unsolved problem: how to share it?

symmetric encryption properties (1)

our functions:

     encrypt: $E(\text{key}, \text{message}) = \text{ciphertext}$
     decrypt: $D(\text{key}, \text{ciphertext}) = \text{message}$

knowing E and D, it should be hard to
learn anything about the message from the ciphertext without key

"hard" $\approx$ would have to try every possible key

symmetric encryption properties (1)

our functions:

  encrypt: $E(\text{key}, \text{message}) = \text{ciphertext}$

  decrypt: $D(\text{key}, \text{ciphertext}) = \text{message}$

knowing E and D, it should be hard to
learn anything about the message from the ciphertext without key

"hard" $\approx$ would have to try every possible key

secrecy properties

actually that's not secret enough, usually want to resist
recovery of info about message or key even given...

partial info about the message, or

lots of other (message, ciphertext) pairs, or
"known plaintext"

lots of (message, ciphertext) pairs for *other messages the attacker
chooses*, or
"chosen plaintext"

lots of (message, ciphertext) pairs encrypted under similar keys, or
"related key"

secrecy properties

actually that's not secret enough, usually want to resist recovery of info about message or key even given…

partial info about the message, or

lots of other (message, ciphertext) pairs, or
  "known plaintext"

lots of (message, ciphertext) pairs for *other messages the attacker chooses*, or
  "chosen plaintext"

lots of (message, ciphertext) pairs encrypted under similar keys, or
  "related key"

secrecy properties

actually that's not secret enough, usually want to resist recovery of info about message or key even given…

partial info about the message, or

lots of other (message, ciphertext) pairs, or
  "known plaintext"

lots of (message, ciphertext) pairs for *other messages the attacker chooses*, or
  "chosen plaintext"

lots of (message, ciphertext) pairs encrypted under similar keys, or
  "related key"

secrecy properties

actually that's not secret enough, usually want to resist recovery of info about message or key even given…

partial info about the message, or

lots of other (message, ciphertext) pairs, or
  "known plaintext"

lots of (message, ciphertext) pairs for *other messages the attacker chooses*, or
  "chosen plaintext"

lots of (message, ciphertext) pairs encrypted under similar keys, or
  "related key"

secrecy properties

actually that's not secret enough, usually want to resist recovery of info about message or key even given...

partial info about the message, or

lots of other (message, ciphertext) pairs, or
　　"known plaintext"

lots of (message, ciphertext) pairs for *other messages the attacker chooses*, or
　　"chosen plaintext"

lots of (message, ciphertext) pairs encrypted under similar keys, or
　　"related key"

using?

in advance: A and B share encryption key

A computes E(key, 'The secret formula is…') = ***

send on network:
A $\rightarrow$ B: ***

using?

in advance: A and B share encryption key

A computes $E(\text{key}, \text{‘The secret formula is…’}) = $ ***

send on network:
A $\rightarrow$ B: ***

B computes $D(\text{key}, \text{***}) = $ ‘The secret formula is …’

encryption is not enough

if B receives an encrypted message from A, and...

it makes sense when decrypted, why isn't that good enough?

problem: an active attacker M
can *selectively* manipulate the encrypted message

manipulating encrypted data?

one example: common symmetric encryption approach:

    use random number + shared secret to…

    produce sequence of hard-to-guess bits x_i as long as the message

    produce ciphertext with xor: $c_i = m_i \oplus x_i$

    message $= m_0 m_1 m_2 \ldots$; ciphertext $= [\text{random number}]c_0 c_1 c_2 \ldots$

means that flipping c_i flips bit m_i

also means that we can shorten messages silently

manipulating messages

as an active attacker

if we know part of plaintext
can sometimes make it read anything else by flipping bits
  "Pay \$100 to Bob" → "Pay \$999 to Bob"

we can shorten
  "Pay \$100 to ABC Corp if they …" → "Pay \$100 to ABC Corp"

we can corrupt selected parts of message and check the response is
  e.g. what changes don't make B reject message as malformed?

message authentication codes (MACs)

goal: use shared secret *key* to verify message origin

one function: $MAC(\text{key}, \text{message}) = \text{tag}$

knowing MAC and the message and the tag, it should be hard to:
    find the value of $MAC(\text{key}, \text{other message})$ — ("forge" the tag)
    find the key

contrast: MAC v checksum

message authentication code acts like checksum, but...

checksum can be recomputed without any key

checksum meant to protect against accidents, not malicious attacks

    checksum can be faster to compute + shorter

using without encryption?

in advance: choose + share MAC key

A prepares message:
    A computes 'Please pay \$100 to M.'
    A computes MAC(MAC key, 'Please pay \$100 to M.') = @@@

A → B: Please pay \$100 to M. @@@

using without encryption?

in advance: choose + share MAC key

A prepares message:
    A computes 'Please pay \$100 to M.'
    A computes MAC(MAC key, 'Please pay \$100 to M.') = @@@

A $\rightarrow$ B: Please pay \$100 to M. @@@

B processes message:
    B recomputes MAC(MAC key, 'Please pay \$100 to M.')
    rejects if it doesn't match @@@

using with encryption?

in advance: choose $+$ share encryption key and MAC key

A prepares message:

    A computes E(encrypt key, 'The secret formula is…') = ***

    A computes MAC(MAC key, ***) = @@@

$A \rightarrow B$: *** @@@

using with encryption?

in advance: choose + share encryption key and MAC key

A prepares message:
> A computes E(encrypt key, 'The secret formula is…') = ***
> A computes MAC(MAC key, ***) = @@@

A → B: *** @@@

B processes message:
> B recomputes MAC(MAC key, ***)
> rejects if it doesn't match @@@
> B computes D(key, ***) = 'The secret formula is …'

"authenticated encryption"

often encryption + MAC packaged together

name: authenticated encryption

exercise

suppose A, B have shared keys K_1, K_2
  assume attackers do not have keys

E/D = encrypt/decrypt function

A asks B to pay Sue $100 by sending message with these parts:
  "2023-11-03: pay $100"
  $E(K_1, \text{"2023-11-03 Sue"})$
  $MAC(K_2, \text{"2023-11-03 \$100"})$

1. can eavesdropper learn: (a) who is being paid, (b) how much?

2. can machine-in-middle change: (a) who is being paid, (b) how much?

shared secrets impractical

problem: shared secrets usually aren't practical

need secure communication before I can do secure communication?

scaling problems
    millions of websites $\times$ billions of browsers $=$ how many keys?
    hard to talk to new people

shared secrets impractical

problem: shared secrets usually aren't practical

need secure communication before I can do secure communication?

scaling problems
    millions of websites $\times$ billions of browsers $=$ how many keys?
    hard to talk to new people

shared secrets impractical

problem: shared secrets usually aren't practical

need secure communication before I can do secure communication?

scaling problems

millions of websites $\times$ billions of browsers $=$ how many keys?

hard to talk to new people

bootstrapping keys?

will still need to have some sort of secure communication to setup!

because we need some way to know we aren't talking to attacker

bootstrapping keys?

will still need to have some sort of secure communication to setup!

because we need some way to know we aren't talking to attacker

but…

bootstrapping keys?

will still need to have some sort of secure communication to setup!

because we need some way to know we aren't talking to attacker

but...

can be broadcast communication
    don't need full new sets of keys for each web browser

bootstrapping keys?

will still need to have some sort of secure communication to setup!

because we need some way to know we aren't talking to attacker

but…

can be broadcast communication
    don't need full new sets of keys for each web browser

only with smaller number of trusted authorities
    don't need to have keys for every website in advance

asymmetric encryption

we'll have two functions:

encrypt: $PE(\text{public key}, \text{message}) = \text{ciphertext}$

decrypt: $PD(\text{private key}, \text{ciphertext}) = \text{message}$

(public key, private key) = "key pair"

key pairs

'private key' = kept secret
   usually not shared with *anyone*

'public key' = safe to give to everyone
   usually some hard-to-reverse function of public key

concept will appear in some other cryptographic primitives

asymmetric encryption properties

functions:

encrypt: $PE(\text{public key}, \text{message}) = \text{ciphertext}$
decrypt: $PD(\text{private key}, \text{ciphertext}) = \text{message}$

should have:

knowing PE, PD, the public key, and ciphertext shouldn't make it too easy to find message

knowing PE, PD, the public key, ciphertext, and message shouldn't help in finding private key

secrecy properties with asymmetric

not going to be able to make things as hard as "try every possibly private key"

but going to make it impractical

like with symmetric encryption want to prevent recovery of *any info about message*

also have some other attacks to worry about:
> e.g. no info about key should be revealed based on our reactions to decrypting maliciously chosen ciphertexts

using asymmetric v symmetric

both:
  use secret data to generate key(s)

asymmetric (AKA public-key) encryption
  one "keypair" per recipient
  private key kept by recipient
  public key sent to all potential senders
  encryption is one-way without private key

symmetric encryption
  one key per (recipient + sender)
  secret key kept by recipient + sender
  if you can encrypt, you can decrypt

using?

in advance: B generates private key + public key

in advance: B sends public key to A (and maybe others) securely

A computes PE(public key, 'The secret formula is…') = *******

send on network:
A $\rightarrow$ B: ********

B computes PD(private key, *******) = 'The secret formula is …'

digital signatures

symmetric encryption : asymetric encryption ::
message authentication codes : digital signatures

digital signatures

pair of functions:

   sign: $S(\text{private key}, \text{message}) = \text{signature}$
   verify: $V(\text{public key}, \text{signature}, \text{message}) = 1$ ("yes, correct signature")

(public key, private key) = key pair (similar to asymmetric encryption)

   public key can be shared with everyone
   knowing S, V, public key, message, signature
   doesn't make it too easy to find another message + signature so that
   $V(\text{public key}, \text{other message}, \text{other signature}) = 1$

using?

in advance: A generates private key + public key

in advance: A sends public key to B (and maybe others) securely

A computes S(private key, 'Please pay ...') = *******

send on network:
A $\rightarrow$ B: 'I authorize the payment', ********

B computes V(public key, 'Please pay ...', *******) = 1

tools, but...

have building blocks, but less than straightforward to use

lots of issues from using building blocks poorly

start of art solution: formal proof sytems

replay attacks

A→B: Did you order lunch? [signature 1 by A]
    signature 1 by A = Sign(A's private signing key, "Did you order lunch?")
    will check with Verify(A's public key, signature 1 by A, "Did you order
    lunch?")

B→A: Yes. [signature 1 by B]
    signature 1 by B = Sign(B's private key, "Yes.")
    will check with Verify(B's public key, signature 1 by B, "Yes.")

A→B: Vegetarian? [signature 2 by A]
B→A: No, not this time. [signature 2 by B]
…
A→B: There's a guy at the door, says he's here to repair the AC.
Should I let him in? [signature N by A]

replay attacks

A→B: Did you order lunch? [signature 1 by A]
B→A: Yes. [signature 1 by B]
A→B: Vegetarian? [signature 2 by A]
B→A: No, not this time. [signature 2 by B]
…
A→B: There's a guy at the door, says he's here to repair the AC.
Should I let him in? [signature ? by A]

how can attacker hijack the reponse to A's inquiry?

replay attacks

A→B: Did you order lunch? [signature 1 by A]
B→A: Yes. [signature 1 by B]
A→B: Vegetarian? [signature 2 by A]
B→A: No, not this time. [signature 2 by B]
…
A→B: There's a guy at the door, says he's here to repair the AC.
Should I let him in? [signature ? by A]

how can attacker hijack the reponse to A's inquiry?

as an attacker, I can copy/paste B's earlier message!
    just keep the same signature, so it can be verified!
    Verify(B's public key, "Yes.", signature 2 from B) = 1

nonces (1)

one solution to replay attacks:

A→B: #1 Did you order lunch? [signature 1 from A]
    signature from A = Sign(A's private key, "#1 Did you order lunch?")

B→A: #1 Yes. [signature 1 from B]
A→B: #2 Vegetarian? [signature 2 from A]
B→A: #2 No, not this time. [signature 2 from B]
…
A→B: #54 There's a guy at the door, says he's here to repair the AC. Should I let him in? [signature ? from A]

(assuming A actually checks the numbers)

nonces (2)

another solution to replay attacks:

B→A: [next number #91523] [signature from B]
A→B: #91523 Did you order lunch? [next number #90382]
[signature from A]
B→A: #90382 Yes. [next number #14578] [signature from B]
…
A→B: #6824 There's a guy at the door, says he's here to repair
the AC. Should I let him in? [next number #36129][signature from
A]

(assuming A actually checks the numbers)

replay attacks (alt)

M→B: #50 Did you order lunch? [signature by M]
B→M: #50 Yes. [signature intended for M by B]

A→B: #50 There's a guy at the door, says he's here to repair the AC. Should I let him in? [signature ? by A]

how can M hijack the reponse to A's inquiry?

replay attacks (alt)

M→B: #50 Did you order lunch? [signature by M]
B→M: #50 Yes. [signature intended for M by B]

A→B: #50 There's a guy at the door, says he's here to repair the AC. Should I let him in? [signature ? by A]

how can M hijack the reponse to A's inquiry?

as an attacker, I can copy/paste B's earlier message!
    just keep the same signature, so it can be verified!
    Verify(B's public key, "#50 Yes.", signature intended for M by B) = 1

confusion about who's sending?

in addition to nonces, either

    write down more who is sending + other context so message can't be reused and/or

    use unique set of keys for each principal you're talking to

with symmetric encryption, also "reflection attacks"

    A sends message to B, attacker sends A's message back to A as if it's from B

other attacks without breaking math

TLS state machine attack

from https://mitls.org/pages/attacks/SMACK

protocol:

    step 1: verify server identity

    step 2: receive messages from server

attack:

    if server sends "here's your next message",

    instead of "here's my identity"

    then broken client ignores verifying server's identity

Matrix vulnerabilties

one example from `https://nebuchadnezzar-megolm.github.io/static/paper.pdf`

system for confidential multi-user chat

protocol + goals:
    each device (my phone, my desktop) has public key
    to talk to me, you verify one of my public keys
    to add devices, my client can forward my other devices' public keys

bug:
    when receiving new keys, clients did not check who they were forwarded
    from correctly

on the lab

getting public keys?

browser talking to websites
needs public keys of every single website?

not really feasible, but…

certificate idea

let's say A has B's public key already.

if C wants B's public key and knows A's already:

A can generate "certificate" for B:
    "B's public key is XXX" AND
    Sign(A's private key, "B's public key is XXX")

B send copy of their "certificate" to C (most common idea)

if C trusts A, now C has B's public key
    if C does not trust A, well, can't trust this either

certificate idea

let's say A has B's public key already.

if C wants B's public key and knows A's already:

A can generate "certificate" for B:
    "B's public key is XXX" AND
    Sign(A's private key, "B's public key is XXX")

B send copy of their "certificate" to C (most common idea)

if C trusts A, now C has B's public key
    if C does not trust A, well, can't trust this either

certificate idea

let's say A has B's public key already.

if C wants B's public key and knows A's already:

A can generate "certificate" for B:
    "B's public key is XXX" AND
    Sign(A's private key, "B's public key is XXX")

B send copy of their "certificate" to C (most common idea)

if C trusts A, now C has B's public key
    if C does not trust A, well, can't trust this either

certificate authorities

websites (and others) go to *certificates authorities* with their public key

certificate authorities sign messages like:
"The public key for foo.com is XXX."

signed message called *certificate*

send certificates to browsers to verify identity

example web certificate (1)

```
Version: 3 (0x2)
Serial Number: 7b:df:f6:ae:2e:d7:db:74:d3:c5:77:ac:bc:44:bf:1b
Signature Algorithm: sha256WithRSAEncryption
Issuer:
    countryName              = US
    stateOrProvinceName      = MI
    localityName             = Ann Arbor
    organizationName         = Internet2
    organizationalUnitName   = InCommon
    commonName               = InCommon RSA Server CA
Validity
    Not Before: Apr 25 00:00:00 2023 GMT
    Not After : Apr 24 23:59:59 2024 GMT
Subject:
    countryName              = US
    stateOrProvinceName      = Virginia
    organizationName         = University of Virginia
    commonName               = canvas.its.virginia.edu
....
X509v3 extensions:
....
    X509v3 Subject Alternative Name: DNS:canvas.its.virginia.edu
```

example web certificate (2)

```
....
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            RSA Public-Key: (2048 bit)
            Modulus:
                00:a2:fb:5a:fb:2d:d2:a7:75:7e:eb:f4:e4:d4:6c:
                94:be:91:a8:6a:21:43:b2:d5:9a:48:b0:64:d9:f7:
                f1:88:fa:50:cf:d0:f3:3d:8b:cc:95:f6:46:4b:42:

....
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
    24:3a:67:c8:0d:ef:eb:8c:eb:ba:8f:d5:11:d2:1e:ea:44:eb:
    fe:af:93:7d:d9:4a:2b:44:a3:7f:47:50:aa:d1:b3:9c:a8:a8:
....
```

certificate chains

That certificate signed by "InCommon RSA Server CA"

CA = certificate authority

so their public key, comes with my OS/browser?
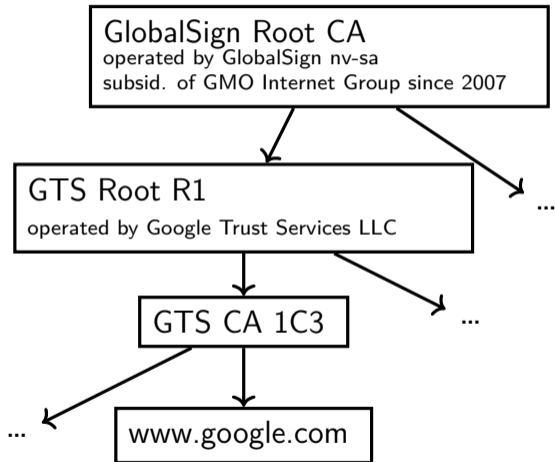   not exactly…

they have their own certificate signed by "USERTrust RSA Certification Authority"

and their public key comes with your OS/browser?

(but both CAs now operated by UK-based Sectigo)

certificate hierarchy

USERTrust RSA
Certification Authority
originally operated by USERTrust, Inc.
acquired by Comodo, Inc (2004)
Comodo's CA division renamed Sectigo (2018)

InCommon
RSA Server CA
operated by Sectigo
on behalf of the Internet2 (not-for-profit)

...

collab.its.virginia.edu

...

GlobalSign Root CA
operated by GlobalSign nv-sa
subsid. of GMO Internet Group since 2007

GTS Root R1
operated by Google Trust Services LLC

...

GTS CA 1C3

...

...

www.google.com

certificate hierarchy

USERTrust RSA
Certification Authority
originally operated by USERTrust, Inc.
acquired by Comodo, Inc (2004)
Comodo's CA division renamed Sectigo (2018)

GlobalSign Root CA
operated by GlobalSign nv-sa
subsid. of GMO Internet Group since 2007

GTS Root R1
operated by Google Trust Services LLC

...

InCommon
RSA Server CA
operated by Sectigo
on behalf of the Internet2 (not-for-profit)

...

GTS CA 1C3

...

...

some "trust anchors" included with browsers and OSes
(for GTS Root R1, only more recent browsers/OSes)

how many trust anchors?

Mozilla Firefox (as of 27 Feb 2023)
    155 trust anchors
    operated by 55 distinct entities

Microsoft Windows (as of 27 Feb 2023)
    237 trust anchors
    operated by 86 distinct entities

public-key infrastructure

ecosystem with certificate authorities
and certificates for everyone

called "public-key infrastructure"

several of these:
 for verifying identity of websites
 for verifying origin of domain name records (kind-of)
 for verifying origin of applications in some OSes/app stores/etc.
 for encrypted email in some organizations
 …

exercise

exercise: how should website certificates verify identity?

how do certificate authorities verify

for web sites, set by CA/Browser Forum

organization of:
    everyone who ships code with list of valid certificate authorities
        Apple, Google, Microsoft, Mozilla, Opera, Cisco, Qihoo 360, Brave, …
    certificate authorities

decide on rules ("baseline requirements") for what CAs do

BR domain name identity validation

options involve CA choosing random value and:

sending it to domain contact (with domain registrar) and receive response with it, or

observing it placed in DNS or website or sent from server in other specific way

exercise: problems this doesn't deal with?

some other things public CAs do

keep their private keys in tamper-resistant hardware

maintain publicly-accessible database of *revoked* certificates
> some browsers check these, sometimes

certificate transparency
> public logs of every certificate issued
> some browsers reject non-logged certificates
> so you can tell if bad certificate exists for your website

'CAA' records in the domain name system
> can indicate which CAs are allowed to issue certificates in DNS
> (but CAs apparently not required to use DNSSEC (certificate
> infrastructure for signing domain name records) when looking this up)

some other things public CAs do

keep their private keys in tamper-resistant hardware

maintain publicly-accessible database of *revoked* certificates
> some browsers check these, sometimes

certificate transparency
> public logs of every certificate issued
> some browsers reject non-logged certificates
> so you can tell if bad certificate exists for your website

'CAA' records in the domain name system
> can indicate which CAs are allowed to issue certificates in DNS
> (but CAs apparently not required to use DNSSEC (certificate
> infrastructure for signing domain name records) when looking this up)

some other things public CAs do

keep their private keys in tamper-resistant hardware

maintain publicly-accessible database of *revoked* certificates
    some browsers check these, sometimes

certificate transparency
    public logs of every certificate issued
    some browsers reject non-logged certificates
    so you can tell if bad certificate exists for your website

'CAA' records in the domain name system
    can indicate which CAs are allowed to issue certificates in DNS
    (but CAs apparently not required to use DNSSEC (certificate
    infrastructure for signing domain name records) when looking this up)

some other things public CAs do

keep their private keys in tamper-resistant hardware

maintain publicly-accessible database of *revoked* certificates
    some browsers check these, sometimes

certificate transparency
    public logs of every certificate issued
    some browsers reject non-logged certificates
    so you can tell if bad certificate exists for your website

'CAA' records in the domain name system
    can indicate which CAs are allowed to issue certificates in DNS
    (but CAs apparently not required to use DNSSEC (certificate
    infrastructure for signing domain name records) when looking this up)

additional crypto tools

cryptographic hash functions (summarize data)

'secure' random numbers

key agreement

motivation: summary for signature

digital signatures typically have size limit

...but we want to sign very large messages

solution: get secure "summary" of message

cryptographic hash

hash(M) = X

given X:
  hard to find message other than by guessing

given X, M:
  hard to find second message so that hash(second message) = X

example uses:
  substitute for original message in digital signature
  building message authentication codes

password hashing

cryptographic hash functions need (basically) guessing to 'reverse'

idea: store cryptographic hash of password instead of password
    attacker who gets hash doesn't get password
    but can still check entered password is correct

password hashing

cryptographic hash functions need (basically) guessing to 'reverse'

idea: store cryptographic hash of password instead of password
    attacker who gets hash doesn't get password
    but can still check entered password is correct

problem: with fast hash function, can try lots of guesses fast

password hashing

cryptographic hash functions need (basically) guessing to 'reverse'

idea: store cryptographic hash of password instead of password
    attacker who gets hash doesn't get password
    but can still check entered password is correct

problem: with fast hash function, can try lots of guesses fast

fix: special slow/resource-intensive cryptograph hash functions
    Argon2i
    scrypt
    PBKDF2

random numbers

need a lot of keys that no one else knows

common task: choose a *random* number

question: what does *random* mean here?

cryptographically secure random numbers

security properties we might want for random numbers:

attacker cannot guess (part of) number better than chance

knowing prior 'random' numbers shouldn't help predict next 'random' numbers

compromising machine now shouldn't reveal older random numbers

exercise: how to generate?

/dev/urandom

Linux kernel random number generator

collects "entropy" from hard-to-predict events
    e.g. exact timing of I/O interrupts
    e.g. some processor's built-in random number circuit

turned into as many random bytes as you want

turning 'entropy' into random bytes

lots of ways to do this; one (rough/incomplete) idea:

internal variable *state*

to add 'entropy'
    state ← SecureHash(state + entropy)

to extract value:
    random bytes ← SecureHash(1 + state)
    give bytes that can't be reversed to compute state

    state ← SecureHash(2 + state)
    change state so attacker can't take us back to old state if compromised

just asymmetric?

given public-key encryption $+$ digital signatures...

why bother with the symmetric stuff?

symmetric stuff much faster

symmetric stuff much better at supporting larger messages

key agreement

problem: A has B's public encryption key
wants to choose shared secret

some ideas:
    A chooses a key, sends it encrypted to B
    A sends a public key encrypted B, B chooses a key and sends it back

key agreement

problem: A has B's public encryption key
wants to choose shared secret

some ideas:

    A chooses a key, sends it encrypted to B

    A sends a public key encrypted B, B chooses a key and sends it back

alternate model:

    both sides generate random values

    derive public-key like "key shares" from values

    use math to combine "key shares"

    kinda like A + B both sending each other public encryption keys

Diffie-Hellman key agreement (2)

A and B want to agree on shared secret

A chooses random value Y

A sends public value derived from Y ("key share")

B chooses random value Z

B sends public value derived from Z ("key share")

A combines Y with public value from B to get number

B combines Z with public value from A to get number
  and b/c of math chosen, both get same number

Diffie-Hellman key agreement (1)

math requirement:

some f, so $f(f(X,Y),Z) = f(f(X,Z),Y)$
(that's hard to invert, etc.)

choose X in advance and:

A randomly chooses Y	B randomly chooses Z
A sends $f(X,Y)$ to B	B sends $f(X,Z)$ to A
A computes $f(f(X,Z),Y)$	B computes $f(f(X,Y),Z)$

key agreement and asym. encryption

can construct public-key encryption from key agreeement

private key: generated random value Y

public key: key share generated from that Y

key agreement and asym. encryption

can construct public-key encryption from key agreeement

private key: generated random value Y

public key: key share generated from that Y

PE(public key, message) =
    generate random value Z
    combine with public key to get shared secret
    use symmetric encryption + MAC using shared secret as keys
    output: (key share generated from Z) (sym. encrypted data) (mac tag)

key agreement and asym. encryption

can construct public-key encryption from key agreeement

private key: generated random value Y

public key: key share generated from that Y

PE(public key, message) =
    generate random value Z
    combine with public key to get shared secret
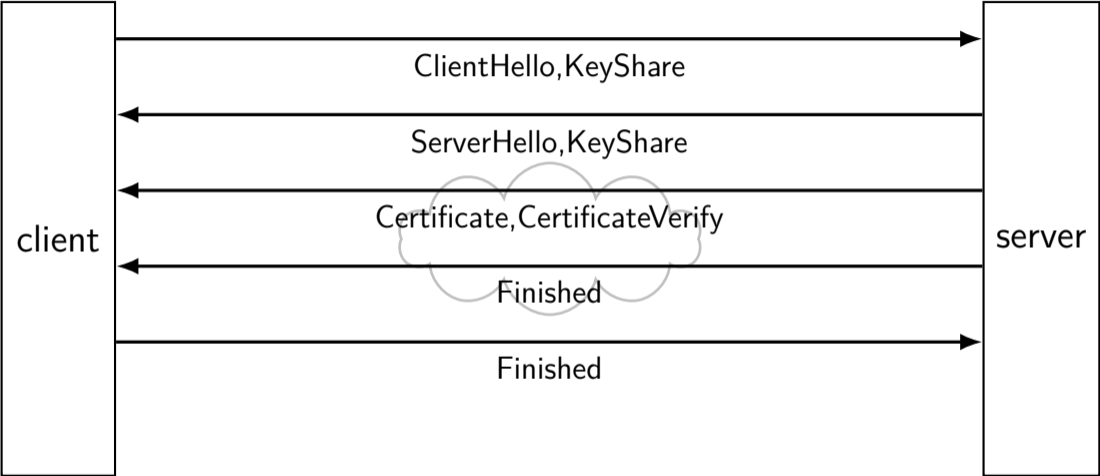   use symmetric encryption + MAC using shared secret as keys
    output: (key share generated from Z) (sym. encrypted data) (mac tag)
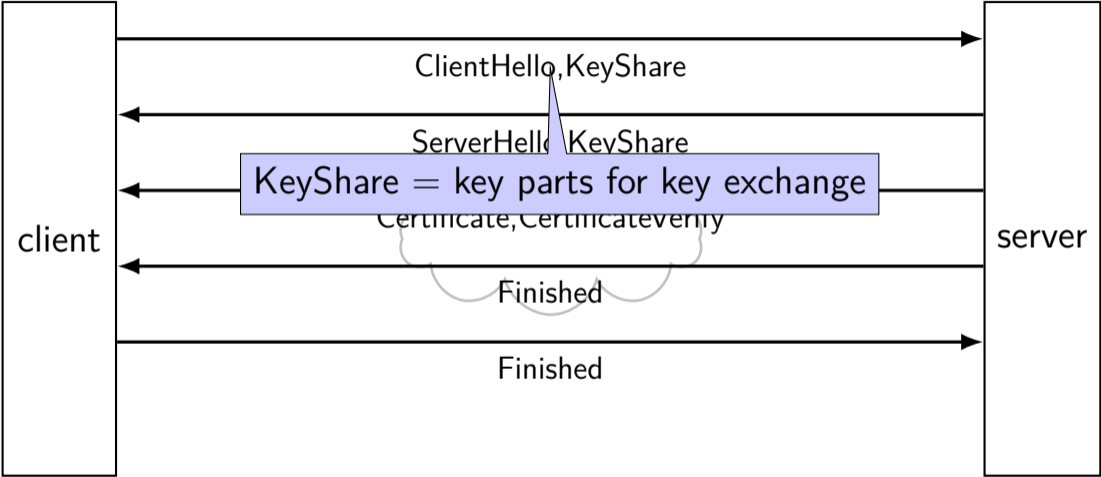
PD(private key, message) =
    extract (key share generated from Z)
    combine with private key to get shared secret, ...

typical TLS handshake

typical TLS handshake

client → server: ClientHello,KeyShare

server → client: ServerHello,KeyShare

KeyShare = key parts for key exchange

server → client: Certificate,CertificateVerify

Finished

client → server: Finished

typical TLS handshake

client → server: ClientHello,KeyShare

server → client: ServerHello,KeyShare

server → client: Certificate,CertificateVerify

server → client: Finished

Certificate = certificate ("foo.com's public key is X" + CA signature)
CertificateVerify = Sign(foo.com's private key, server's key share)

typical TLS handshake

client → server: ClientHello,KeyShare

server → client: ServerHello,KeyShare

server → client: Certificate,CertificateVerify

server → client: Finished

client → server: Finished

MAC(key made from key shares, Hash(everything so far)) (purpose: tie new key with rest of handshake)

typical TLS handshake

client → server: ClientHello,KeyShare

server → client: ServerHello,KeyShare

server → client: Certificate,CertificateVerify

server → client: Finished

client → server: Finished
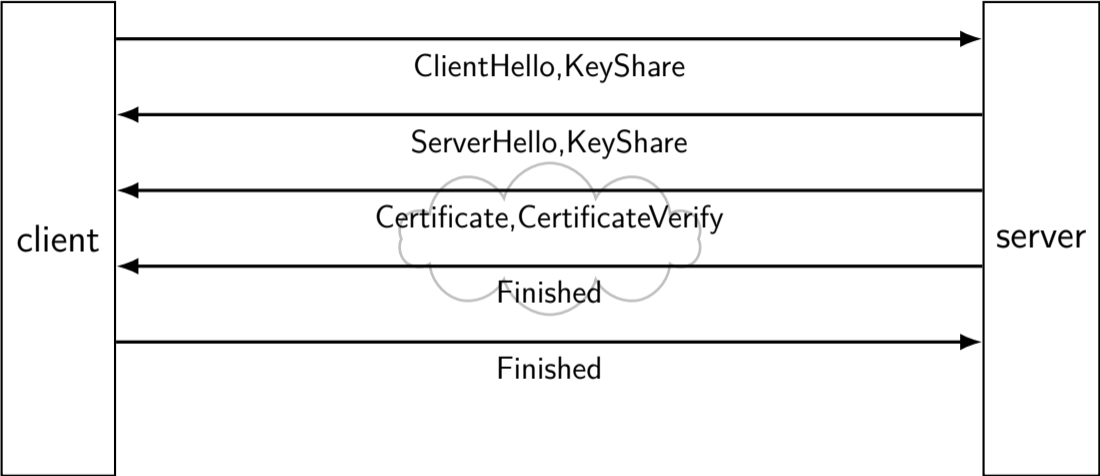
MAC(key made from key shares, Hash(everything so far))
(purpose: tie new key with rest of handshake)

typical TLS handshake

client → server: ClientHello,KeyShare

server → client: ServerHello,KeyShare

server → client: Certificate,CertificateVerify

server → client: Finished

client → server: Finished

MAC(key made from key shares, Hash(everything so far))
(purpose: tie new key with rest of handshake)

typical TLS handshake

TLS: after handshake

use key shares results to get **several** keys
    take hash(something + shared secret) to derive each key

separate keys for each direction (server $\rightarrow$ client and vice-versa)

often separate keys for encryption and MAC

later messages use encryption + MAC + nonces

things modern TLS usually does

(not all these properties provided by all TLS versions and modes)

confidentiality/authenticity
    server = one ID'd by certificate
    client = same throughout whole connection

forward secrecy
    can't decrypt old conversations (data for KeyShares is temporary)

fast
    most communication done with more efficient symmetric ciphers
    1 set of messages back and forth to setup connection

denial of service (1)

so far: worried about network attacker disrupting confidentiality/authenticity

what if we're just worried about just breaking things

well, if they control network, nothing we can do...

but often worried about less

denial of service (2)

if you just want to inconvenience...

attacker just sends lots of stuff to my server

my server becomes overloaded?

my network becomes overloaded?

but: doesn't this require a lot of work for attacker?

exercise: why is this often not a big obstacle

denial of service: asymmetry

work for attacker > work for defender

how much computation per message?
    complex search query?
    something that needs tons of memory?
    something that needs to read tons from disk?

how much sent back per message?

resources for attacker > resources of defender

how many machines can attacker use?

denial of service: reflection/amplification

instead of sending messages directly…attacker can send messages "from" you to third-party

third-party sends back replies that overwhelm network

example: short DNS query with lots of things in response

"amplification" =
   third-party inadvertantly turns small attack into big one

firewalls

don't want to expose network service to everyone?

solutions:
    service picky about who it accepts connections from
    filters in OS on machine with services
    filters on router

later two called "firewalls"

firewall rules examples?

ALLOW tcp port 443 (https) FROM everyone

ALLOW tcp port 22 (ssh) FROM my desktop's IP address

BLOCK tcp port 22 (ssh) FROM everyone else

ALLOW from address X to address Y

…

network security summary (1)

communicating securely with math
    secret value (shared key, public key) that attacker can't have
    symmetric: shared keys used for (de)encryption + auth/verify; fast
    asymmetric: public key used by any for encrypt + verify; slower
    asymmetric: private key used by holder for decrypt + sign; slower

protocol attacks — repurposing encrypt/signed/etc. messages

certificates — verifiable forwarded public keys

key agreement — for generated shared-secret "in public"
    publish key shares from private data
    combine private data with key share for shared secret

network security summary (2)

TLS: combine all cryptography stuff to make "secure channel"

denial-of-service — attacker just disrupts/overloads (not subtle)

firewalls

backup slides

backup slides

URLs and HTTP (1)

`http://www.foo.com:80/foo/bar?quux#q1`

lookup IP address of www.foo.com

connect via TCP to port 80:
```
GET /foo/bar?quux HTTP/1.1
Host: www.foo.com:80
```

URLs and HTTP (1)

`http://www.foo.com:80/foo/bar?quux``#q1`

lookup IP address of www.foo.com

connect via TCP to port 80:
```
GET /foo/bar?quux HTTP/1.1
Host: www.foo.com:80
```

URLs and HTTP (1)

```
http://www.foo.com:80/foo/bar?quux#q1
```

lookup IP address of www.foo.com

connect via TCP to port 80:
```
GET /foo/bar?quux HTTP/1.1
Host: www.foo.com:80
```
exercise: why include the Host there?

spoofing

if I only allow connections from my desktop's IP addresses,
how would you attack this?

hint: how do we know what address messages come from?

link layer reliablity?

Ethernet + Wifi have checksums

Q1: Why doesn't this give us uncorrupted messages?
   Why do we still have checksums at the higher layers?

Q2: What's a benefit of doing this if we're also doing it in the
higher layer?

link layer quality of service

if frame gets...

event	on Ethernet	on WiFi
collides with another	detected + may resend	resend
not received	lose silently	resent
header corrupted	usually discard silently	usually resend
data corrupted	usually discard silently	usually resend
too long	not allowed to send	not allowed to send
reordered (v. other messages)	received out of order	received out of order
destination unknown	lose silently	usually resend??
too much being sent	discard excess?	discard excess?

network layer quality of service

if packet …

event	on IPv4/v6
collides with another	out of scope — handled by link layer
not received	lost silently
header corrupted	usually discarded silently
data corrupted	received corrupted
too long	dropped with notice or "fragmented" + recombined
reordered (v. other messages)	received out of order
destination unknown	usually dropped with notice
too much being sent	discard excess

network layer quality of service

if packet …

event	on IPv4/v6
collides with another	out of scope — handled by link layer
not received	lost silently
header corrupted	usually discarded silently
data corrupted	received corrupted
too long	dropped with notice or "fragmented" + recombined
reordered (v. other messages)	received out of order
destination unknown	usually dropped with notice
too much being sent	discard excess

includes dropped by link layer
(e.g. if detected corrupted there)

firewalls

don't want to expose network service to everyone?

solutions:
    service picky about who it accepts connections from
    filters in OS on machine with services
    filters on router

later two called "firewalls"

firewall rules examples?

ALLOW tcp port 443 (https) FROM everyone

ALLOW tcp port 22 (ssh) FROM my desktop's IP address

BLOCK tcp port 22 (ssh) FROM everyone else

ALLOW from address X to address Y

…

t

querying the root

```
$ dig +trace +all www.cs.virginia.edu
...
edu.                    172800      IN      NS      b.edu-servers.net.
edu.                    172800      IN      NS      f.edu-servers.net.
edu.                    172800      IN      NS      i.edu-servers.net.
edu.                    172800      IN      NS      a.edu-servers.net.
...
b.edu-servers.net.      172800      IN      A       191.33.14.30
b.edu-servers.net.      172800      IN      AAAA    2001:503:231d::2:30
f.edu-servers.net.      172800      IN      A       192.35.51.30
f.edu-servers.net.      172800      IN      AAAA    2001:503:d414::30
...
;; Received 843 bytes from 198.97.190.53#53(h.root-servers.net) in 8 ms
...
```

querying the edu

```
$ dig +trace +all www.cs.virginia.edu
...
virginia.edu.                172800      IN      NS      nom.virginia.edu.
virginia.edu.                172800      IN      NS      uvaarpa.virginia.edu.
virginia.edu.                172800      IN      NS      eip-01-aws.net.virginia.edu.
nom.virginia.edu.       172800      IN      A       128.143.107.101
uvaarpa.virginia.edu.   172800      IN      A       128.143.107.117
eip-01-aws.net.virginia.edu. 172800 IN      A       44.234.207.10
;; Received 165 bytes from 192.26.92.30#53(c.edu-servers.net) in 40 ms
...
```

querying virginia.edu+cs.virginia.edu

```
$ dig +trace +all www.cs.virginia.edu
...
cs.virginia.edu.          3600          IN          NS          coresrv01.cs.virginia.edu.
coresrv01.cs.virginia.edu. 3600          IN          A          128.143.67.11
;; Received 116 bytes from 44.234.207.10#53(eip-01-aws.net.virginia.edu) in 72 ms

www.cs.Virginia.EDU.          172800          IN          A          128.143.67.11
cs.Virginia.EDU.          172800          IN          NS          coresrv01.cs.Virginia.EDU.
coresrv01.cs.Virginia.EDU. 172800 IN          A          128.143.67.11
;; Received 151 bytes from 128.143.67.11#53(coresrv01.cs.virginia.edu) in 4 ms
```

querying typical ISP's resolver

```
$ dig www.cs.virginia.edu
...
;; ANSWER SECTION:
www.cs.Virginia.EDU.          7183          IN          A          128.143.67.11
..
```

cached response

valid for 7183 more seconds

after that everyone needs to check again

'connected' UDP sockets

```c
int fd = socket(AF_INET, SOCK_DGRAM, 0);
struct sockaddr_in my_addr= ...;
/* set local IP address + port */
bind(fd, &my_addr, sizeof(my_addr))
struct sockaddr_in to_addr = ...;
connect(fd, &to_addr); /* set remote IP address + port */
    /* doesn't actually communicate with remote address yet */
...
int count = write(fd, data, data_size);
// OR
int count = send(fd, data, data_size, 0 /* flags */);
    /* single message -- sent ALL AT ONCE */

int count = read(fd, buffer, buffer_size);
// OR
int count = recv(fd, buffer, buffer_size, 0 /* flags */);
    /* receives whole single message ALL AT ONCE */
```

UDP sockets on IPv4

```c
int fd = socket(AF_INET, SOCK_DGRAM, 0);
struct sockaddr_in my_addr= ...;
/* set local IP address + port */
if (0 != bind(fd, &my_addr, sizeof(my_addr)))
    handle_error();
...
struct sockaddr_in to_addr = ...;
    /* send a message to specific address */
int bytes_sent = sendto(fd, data, data_size, 0 /* flags */,
    &to_addr, sizeof(to_addr));

struct sockaddr_in from_addr = ...;
    /* receive a message + learn where it came from */
int bytes_recvd = recvfrom(fd, &buffer[0], buffer_size, 0,
    &from_addr, sizeof(from_addr));
...
```

what about non-local machines?

when configuring network specify:

range of addresses to expect on local network
    128.148.67.0-128.148.67.255 on my desktop
    "netmask"

gateway machine to send to for things outside my local network
    128.143.67.1 on my desktop
    my desktop looks up the corresponding MAC address

routes on my desktop

```
$ /sbin/route -n
Kernel IP routing table
Destination     Gateway         Genmask          Flags Metric Ref    Use Iface
0.0.0.0         128.143.67.1    0.0.0.0          UG    100    0        0 enp0s31f6
128.143.67.0    0.0.0.0         255.255.255.0    U     100    0        0 enp0s31f6
169.254.0.0     0.0.0.0         255.255.0.0      U     1000   0        0 enp0s31f6
```

network configuration says:

(line 2) to get to 128.143.67.0–128.143.67.255, send directly on local network

    "genmask" is mask (for bitwise operations) to specify how big range is

(line 3) to get to 169.254.0.0–169.254.255.255, send directly on local network

(line 1) to get anywhere else, use "gateway" 128.143.67.1

querying the root

```
$ dig +trace +all www.cs.virginia.edu
...
edu.                    172800      IN      NS      b.edu-servers.net.
edu.                    172800      IN      NS      f.edu-servers.net.
edu.                    172800      IN      NS      i.edu-servers.net.
edu.                    172800      IN      NS      a.edu-servers.net.
...
b.edu-servers.net.      172800      IN      A       191.33.14.30
b.edu-servers.net.      172800      IN      AAAA    2001:503:231d::2:30
f.edu-servers.net.      172800      IN      A       192.35.51.30
f.edu-servers.net.      172800      IN      AAAA    2001:503:d414::30
...
;; Received 843 bytes from 198.97.190.53#53(h.root-servers.net) in 8 ms
...
```

querying the edu

```
$ dig +trace +all www.cs.virginia.edu
...
virginia.edu.                 172800        IN      NS      nom.virginia.edu.
virginia.edu.                 172800        IN      NS      uvaarpa.virginia.edu.
virginia.edu.                 172800        IN      NS      eip-01-aws.net.virginia.edu.
nom.virginia.edu.      172800         IN      A       128.143.107.101
uvaarpa.virginia.edu.         172800        IN      A       128.143.107.117
eip-01-aws.net.virginia.edu. 172800 IN        A        44.234.207.10
;; Received 165 bytes from 192.26.92.30#53(c.edu-servers.net) in 40 ms
...
```

querying virginia.edu+cs.virginia.edu

```
$ dig +trace +all www.cs.virginia.edu
...
cs.virginia.edu.          3600         IN         NS        coresrv01.cs.virginia.edu.
coresrv01.cs.virginia.edu. 3600        IN         A         128.143.67.11
;; Received 116 bytes from 44.234.207.10#53(eip-01-aws.net.virginia.edu) in 72 ms

www.cs.Virginia.EDU.          172800         IN         A         128.143.67.11
cs.Virginia.EDU.          172800         IN         NS        coresrv01.cs.Virginia.EDU.
coresrv01.cs.Virginia.EDU. 172800 IN         A         128.143.67.11
;; Received 151 bytes from 128.143.67.11#53(coresrv01.cs.virginia.edu) in 4 ms
```

querying typical ISP's resolver

```
$ dig www.cs.virginia.edu
...
;; ANSWER SECTION:
www.cs.Virginia.EDU.          7183          IN          A          128.143.67.11
..
```

cached response

valid for 7183 more seconds

after that everyone needs to check again

connection setup: server, manual

```
int server_socket_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY; /* "any address I can use" */
    /* or: addr.s_addr.in_addr = INADDR_LOOPBACK (127.0.0.1) */
    /* or: addr.s_addr.in_addr = htonl(...); */
addr.sin_port = htons(9999); /* port number 9999 */

if (bind(server_socket_fd, &addr, sizeof(addr)) < 0) {
    /* handle error */
}
listen(server_socket_fd, MAX_NUM_WAITING);
...
int socket_fd = accept(server_socket_fd, NULL);
```

connection setup: server, manual

```
int server_socket_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY; /* "any address I can use" */
    /* or: addr.s_addr.in_addr = INADDR_LOOPBACK (127.0.0.1) */
    /* or: addr.s_addr.in_addr = htonl(...); */
addr.sin_port = htons(9999); /* port number 9999 */

if (bind(server_socket_fd, &addr, sizeof(addr)) < 0) {
    /* handle error */
}
listen
...
int so
```

INADDR_ANY: accept connections for any address I can!
alternative: specify specific address

connection setup: server, manual

```
int server_socket_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY; /* "any address I can use" */
    /* or: addr.s_addr.in_addr = INADDR_LOOPBACK (127.0.0.1) */
    /* or: addr.s_addr.in_addr = htonl(...); */
addr.sin_port = htons(9999); /* port number 9999 */

if (bind(server_socket_fd, &addr, sizeof(addr)) < 0) {
    /* handle error */
}
lis
...
int
```

bind to 127.0.0.1? only accept connections from same machine
what we recommend for FTP server assignment

connection setup: server, manual

```
int server_socket_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY; /* "any address I can use" */
    /* or: addr.s_addr.in_addr = INADDR_LOOPBACK (127.0.0.1) */
    /* or: addr.s_addr.in_addr = htonl(...); */
addr.sin_port = htons(9999); /* port number 9999 */

if (bind(server_socket_fd, &addr, sizeof(addr)) < 0) {
    /* handle error */
}
listen(serv  choose the number of unaccepted connections
...
int socket_fd = accept(server_socket_fd, NULL);
```

connection setup: client — manual addresses

```
int sock_fd;

server = /* code on later slide */;
sock_fd = socket(
    AF_INET, /* IPv4 */
    SOCK_STREAM, /* byte-oriented */
    IPPROTO_TCP
);
if (sock_fd < 0) { /* handle error */ }

struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(2156872459); /* 128.143.67.11 */
addr.sin_port = htons(80); /* port 80 */
if (connect(sock_fd, (struct sockaddr*) &addr, sizeof(addr)) {
    /* handle error */
}
DoClientStuff(sock_fd); /* read and write from sock_fd */
```

connection setup: client — manual addresses

```
int sock_fd;

server = /* code on later slide */;
sock_fd = socket(
    AF_INET, /* IPv4 */
    SOCK_STREAM, /* byte-oriented */
    IPPROTO_TCP
);
if (sock_fd < 0) { /* handle error */ }
```

specify IPv4 instead of IPv6 or local-only sockets
specify TCP (byte-oriented) instead of UDP ('datagram' oriented)

```
addr.sin_addr.s_addr = htonl(2156872459); /* 128.143.67.11 */
addr.sin_port = htons(80); /* port 80 */
if (connect(sock_fd, (struct sockaddr*) &addr, sizeof(addr)) {
    /* handle error */
}
DoClientStuff(sock_fd); /* read and write from sock_fd */
```

connection setup: client — manual addresses

```
int sock_fd;

server = /* co   htonl/s = host-to-network long/short
sock_fd = socke  network byte order = big endian
    AF_INET, /*
    SOCK_STREAM, /* byte-oriented */
    IPPROTO_TCP
);
if (sock_fd < 0) { /* handle error */ }

struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(2156872459); /* 128.143.67.11 */
addr.sin_port = htons(80); /* port 80 */
if (connect(sock_fd, (struct sockaddr*) &addr, sizeof(addr)) {
    /* handle error */
}
DoClientStuff(sock_fd); /* read and write from sock_fd */
```

connection setup: client — manual addresses

```
int sock_fd;

server = /  struct representing IPv4 address + port number
sock_fd =   declared in <netinet/in.h>
    AF_INE  see man 7 ip on Linux for docs
    SOCK_S
    IPPROTO_TCP
);
if (sock_fd < 0) { /* handle error */ }

struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(2156872459); /* 128.143.67.11 */
addr.sin_port = htons(80); /* port 80 */
if (connect(sock_fd, (struct sockaddr*) &addr, sizeof(addr)) {
    /* handle error */
}
DoClientStuff(sock_fd); /* read and write from sock_fd */
```

echo client/server

```
void client_for_connection(int socket_fd) {
    int n; char send_buf[MAX_SIZE]; char recv_buf[MAX_SIZE];
    while (prompt_for_input(send_buf, MAX_SIZE)) {
        n = write(socket_fd, send_buf, strlen(send_buf));
        if (n != strlen(send_buf)) {...error?...}
        n = read(socket_fd, recv_buf, MAX_SIZE);
        if (n <= 0) return; // error or EOF
        write(STDOUT_FILENO, recv_buf, n);
    }
}
```

```
void server_for_connection(int socket_fd) {
    int read_count, write_count; char request_buf[MAX_SIZE];
    while (1) {
        read_count = read(socket_fd, request_buf, MAX_SIZE);
        if (read_count <= 0) return; // error or EOF
        write_count = write(socket_fd, request_buf, read_count);
        if (read_count != write_count) {...error?...}
    }
```

echo client/server

```
void client_for_connection(int socket_fd) {
    int n; char send_buf[MAX_SIZE]; char recv_buf[MAX_SIZE];
    while (prompt_for_input(send_buf, MAX_SIZE)) {
        n = write(socket_fd, send_buf, strlen(send_buf));
        if (n != strlen(send_buf)) {...error?...}
        n = read(socket_fd, recv_buf, MAX_SIZE);
        if (n <= 0) return; // error or EOF
        write(STDOUT_FILENO, recv_buf, n);
    }
}
```

```
void server_for_connection(int socket_fd) {
    int read_count, write_count; char request_buf[MAX_SIZE];
    while (1) {
        read_count = read(socket_fd, request_buf, MAX_SIZE);
        if (read_count <= 0) return; // error or EOF
        write_count = write(socket_fd, request_buf, read_count);
        if (read_count != write_count) {...error?...}
    }
```

echo client/server

```
void client_for_connection(int socket_fd) {
    int n; char send_buf[MAX_SIZE]; char recv_buf[MAX_SIZE];
    while (prompt_for_input(send_buf, MAX_SIZE)) {
        n = write(socket_fd, send_buf, strlen(send_buf));
        if (n != strlen(send_buf)) {...error?...}
        n = read(socket_fd, recv_buf, MAX_SIZE);
        if (n <= 0) return; // error or EOF
        write(STDOUT_FILENO, recv_buf, n);
    }
}
```

```
void server_for_connection(int socket_fd) {
    int read_count, write_count; char request_buf[MAX_SIZE];
    while (1) {
        read_count = read(socket_fd, request_buf, MAX_SIZE);
        if (read_count <= 0) return; // error or EOF
        write_count = write(socket_fd, request_buf, read_count);
        if (read_count != write_count) {...error?...}
    }
```

connection setup: server, address setup

```c
/* example (hostname, portname) = ("127.0.0.1", "443") */
const char *hostname; const char *portname;
...
struct addrinfo *server;
struct addrinfo hints;
int rv;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET; /* for IPv4 */
/* or: */ hints.ai_family = AF_INET6; /* for IPv6 */
/* or: */ hints.ai_family = AF_UNSPEC; /* I don't care */
hints.ai_flags = AI_PASSIVE;

rv = getaddrinfo(hostname, portname, &hints, &server);
if (rv != 0) { /* handle error */ }
```

connection setup: server, address setup

```
/* example (hostname, portname) = ("127.0.0.1", "443") */
const char *hostname; const char *portname;
...
struct addrinfo *server;
struct addrinfo hints;
int rv;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET; /* for IPv4 */
/* or: */ hints.ai_family = AF_INET6; /* for IPv6 */
/* or: */ hints.ai_family = AF_UNSPEC; /* I don't care */
hints.ai_flags = 
                  hostname could also be NULL
rv = getaddrinfo   means "use all possible addresses"   ver);
if (rv != 0) { /   only makes sense for servers
```

connection setup: server, address setup

```c
/* example (hostname, portname) = ("127.0.0.1", "443") */
const char *hostname; const char *portname;
...
struct addrinfo *server;
struct addrinfo hints;
int rv;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET; /* for IPv4 */
/* or: */ hints.ai_family = AF_INET6; /* for IPv6 */
/* or: */ hints.ai_family = AF_UNSPEC; /* I don't care */
hints.ai_flags
                portname could also be NULL
                means "choose a port number for me"
rv = getaddrin                                      er);
if (rv != 0) {  only makes sense for servers
```

141

connection setup: server, address setup

```
/* example (hostname, portname) = ("127.0.0.1", "443") */
const char *ho
...            AI_PASSIVE: "I'm going to use bind"
struct addrinfo *server;
struct addrinfo hints;
int rv;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET; /* for IPv4 */
/* or: */ hints.ai_family = AF_INET6; /* for IPv6 */
/* or: */ hints.ai_family = AF_UNSPEC; /* I don't care */
hints.ai_flags = AI_PASSIVE;

rv = getaddrinfo(hostname, portname, &hints, &server);
if (rv != 0) { /* handle error */ }
```

connection setup: server, addrinfo

```
struct addrinfo *server;
... getaddrinfo(...) ...

int server_socket_fd = socket(
    server->ai_family,
    server->ai_sockttype,
    server->ai_protocol
);

if (bind(server_socket_fd, ai->ai_addr, ai->ai_addr_len)) < 0) {
    /* handle error */
}
listen(server_socket_fd, MAX_NUM_WAITING);
...
int socket_fd = accept(server_socket_fd, NULL);
```

connection setup: client, using addrinfo

```
int sock_fd;
struct addrinfo *server = /* code on next slide */;

sock_fd = socket(
    server->ai_family,
    // ai_family = AF_INET (IPv4) or AF_INET6 (IPv6) or ...
    server->ai_socktype,
    // ai_socktype = SOCK_STREAM (bytes) or ...
    server->ai_prototcol
    // ai_protocol = IPPROTO_TCP or ...
);
if (sock_fd < 0) { /* handle error */ }
if (connect(sock_fd, server->ai_addr, server->ai_addrlen) < 0) {
    /* handle error */
}
freeaddrinfo(server);
DoClientStuff(sock_fd); /* read and write from sock_fd */
close(sock_fd);
```

connection setup: client, using addrinfo

```
int sock_fd;
struct addrinfo *server = /* code on next slide */;

sock_fd = socket(
    server->ai_family,
    // ai_family = AF_INET (IPv4) or AF_INET6 (IPv6) or ...
    server->ai_socktype,
    // ai_socktype = SOCK_STREAM (bytes) or ...
    server->ai_protocol
    //
);
if (so
if (co                                              < 0) {
    /*
}
freeaddrinfo(server);
DoClientStuff(sock_fd); /* read and write from sock_fd */
close(sock_fd);
```

addrinfo contains all information needed to setup socket
set by `getaddrinfo` function (next slide)
handles IPv4 and IPv6
handles DNS names, service names

connection setup: client, using addrinfo

```
int sock_fd;
struct addrinfo *server = /* code on next slide */;

sock_fd = socket(
    server->ai_family,
    // ai_family = AF_INET (IPv4) or AF_INET6 (IPv6) or ...
    server->ai_socktype,
    // ai_socktype = SOCK_STREAM (bytes) or ...
    server->ai_prototcol
    // ai_protocol = IPPROTO_TCP or ...
);
if (sock_fd < 0) { /* handle error */ }
if (connect(sock_fd, server->ai_addr, server->ai_addrlen) < 0) {
    /* handle error */
}
freeaddrinfo(server);
DoClientStuff(sock_fd); /* read and write from sock_fd */
close(sock_fd);
```

connection setup: client, using addrinfo

```
int sock_fd;
struct addrinfo *server;        /* we have set up client */

sock_fd = socket(
    server->ai_family,
    // ai_family = AF_INET (IPv4) or AF_INET6 (IPv6) or ...
    server->ai_socktype,
    // ai_socktype = SOCK_STREAM (bytes) or ...
    server->ai_prototcol
    // ai_protocol = IPPROTO_TCP or ...
);
if (sock_fd < 0) { /* handle error */ }
if (connect(sock_fd, server->ai_addr, server->ai_addrlen) < 0) {
    /* handle error */
}
freeaddrinfo(server);
DoClientStuff(sock_fd); /* read and write from sock_fd */
close(sock_fd);
```

ai_addr points to struct representing address
type of struct depends whether IPv6 or IPv4

143

connection setup: client, using addrinfo

```
int sock_fd;
st                                                                    ┌──────────────────────────────────────────────────────┐
                                                                      │ since addrinfo contains pointers to dynamically allocated memory, │
so                                                                    │ call this function to free everything                  │
                                                                      └──────────────────────────────────────────────────────┘
        // ai_family = AF_INET (IPv4) or AF_INET6 (IPv6) or ...
        server->ai_socktype,
        // ai_socktype = SOCK_STREAM (bytes) or ...
        server->ai_prototcol
        // ai_protocol = IPPROTO_TCP or ...
);
if (sock_fd < 0) { /* handle error */ }
if (connect(sock_fd, server->ai_addr, server->ai_addrlen) < 0) {
    /* handle error */
}
freeaddrinfo(server);
DoClientStuff(sock_fd); /* read and write from sock_fd */
close(sock_fd);
```

connection setup: lookup address

```
/* example hostname, portname = "www.cs.virginia.edu", "443" */
const char *hostname; const char *portname;
...
struct addrinfo *server;
struct addrinfo hints;
int rv;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC; /* for IPv4 OR IPv6 */
// hints.ai_family = AF_INET4; /* for IPv4 only */

hints.ai_socktype = SOCK_STREAM; /* byte-oriented --- TCP */
rv = getaddrinfo(hostname, portname, &hints, &server);
if (rv != 0) { /* handle error */ }

/* eventually freeaddrinfo(result) */
```

connection setup: lookup address

```
/* example hostname, portname = "www.cs.virginia.edu", "443" */
const char *hostname; const char *portname;
...
struct addrinfo *server;
struct addrinfo hints;
int rv;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC; /* for IPv4 OR IPv6 */
// hints.ai_family = AF_INET4; /* for IPv4 only */
```

NB: pass pointer *to pointer* to addrinfo to fill in

```
hints.ai_socktype = SOCK_STREAM; /* byte-oriented --- TCP */
rv = getaddrinfo(hostname, portname, &hints, &server);
if (rv != 0) { /* handle error */ }

/* eventually freeaddrinfo(result) */
```

connection setup: lookup address

```
/* example hostname, portname = "www.cs.virginia.edu", "443" */
const
...
struct
struct
int rv;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC; /* for IPv4 OR IPv6 */
// hints.ai_family = AF_INET4; /* for IPv4 only */

hints.ai_socktype = SOCK_STREAM; /* byte-oriented --- TCP */
rv = getaddrinfo(hostname, portname, &hints, &server);
if (rv != 0) { /* handle error */ }

/* eventually freeaddrinfo(result) */
```

AF_UNSPEC: choose between IPv4 and IPv6 for me
AF_INET, AF_INET6: choose IPv4 or IPV6 respectively

connection setup: multiple server addresses

```
struct addrinfo *server;
...
rv = getaddrinfo(hostname, portname, &hints, &server);
if (rv != 0) { /* handle error */ }

for (struct addrinfo *current = server; current != NULL;
     current = current->ai_next) {
    sock_fd = socket(current->ai_family, current->ai_socktype, curr
    if (sock_fd < 0) continue;
    if (connect(sock_fd, current->ai_addr, current->ai_addrlen) ==
        break;
    }
    close(sock_fd); // connect failed
}
freeaddrinfo(server);
DoClientStuff(sock_fd);
close(sock_fd);
```

connection setup: multiple server addresses

```
struct addrinfo *server;
...
rv = getaddrinfo(hostname, portname, &hints, &server);
if (rv != 0) { /* handle error */ }

for (struct addrinfo *current = server; current != NULL;
     current = current->ai_next) {
    sock_fd = socket(current->ai_family, current->ai_socktype, curr
    if (sock_fd < 0) continue;
    if (connect(sock_fd, current->ai_addr, current->ai_addrlen) ==
        break;
    }
    clos
}
freeaddr
DoClient
close(s
```

addrinfo is a linked list
name can correspond to multiple addresses
example: redundant copies of web server
example: an IPv4 address and IPv6 address

connection setup: old lookup function

```
/* example hostname, portnum= "www.cs.virginia.edu", 443*/
const char *hostname; int portnum;
...
struct hostent *server_ip;
server_ip = gethostbyname(hostname);

if (server_ip == NULL) { /* handle error */ }

struct sockaddr_in addr;
addr.s_addr = *(struct in_addr*) server_ip->h_addr_list[0];
addr.sin_port = htons(portnum);
sock_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
connect(sock_fd, &addr, sizeof(addr));
...
```

aside: on server port numbers

Unix convention: must be `root` to use ports 0–1023
     root = superuser = 'adminstrator user' = what sudo does

so, for testing: probably ports > 1023