last time

(review of?) sockets

- read()/write() interface
- bind() (set address); listen()+accept()/connect()

UDP sockets

tracking connections → file descriptors

domain name system

- hierarchical database; caching

URIs

mapping link-layer to network-layer addresses

address discovery — DHCP

anonymous feedback — quizzes (1)

"I love this class being my 6th HSS elective. Why are these quizzes more similar to a section from the SAT's reading section than quizzes from a CS class."

"Would we be able to move the quizzes until after the homework and/or labs? I feel I understand the topic much better after those but generally the quizzes are harder to understand since we have little to no experience with them by the point we take them."

- there will be times when we don't do assignment until week+ after lecture coverage
  - (based on my guesses about post-Spring-break scheduling and making sure I have time if lecture goes slowly)
- think I'm getting lots of complaints either way

anonymous feedback — last quiz

"We were given conflicting information for solving the first question on the quiz. One slides suggests that we should multiply the 12 most significant bits of the given 0x1231 with 8(the page size), but the example we did during class suggests we never implement this step. As a result, I had two answers for that quiz question, namely 0x9180 and 0x1230. I feel that partial credit should be given for said question."

the page size in the quiz question was **16 bytes** (= 8 PTEs)

…so the page size to multiply by was 16

in the example in lecture, PTEs were 1 byte —

so as a special case page size in bytes = page size in PTEs

anonymous feedback — OH waits

"In OH [TA name] spent 30+ minutes helping the same student after two new students had written their names on the whiteboard."

anonymous feedback — HW due time

"Can the assignments (not labs) be due at 11:59pm on the day it's due?"

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

NAT and layers

previously: network layer responsible for get to right machine

now: network + transport layer

because we use port numbers

also, NAT needs to know about connections (transport layer)

to know how to setup/remove table entries

secure communication context

"secure" communication

mostly talk about on network

between *principals* $\approx$ 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 (man-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 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 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 M, E do not)

(later: easier to setup assumptions)

bad ways to use shared secret

A $\rightarrow$ B: What's the password?

B $\rightarrow$ A: It's '`Abc$xyM$e`'.

A $\rightarrow$ B: That's right! Here's my confidential information.

bad ways to use shared secret

A $\rightarrow$ B: What's the password?

B $\rightarrow$ A: It's '`Abc$xyM$e`'.

A $\rightarrow$ B: That's right! Here's my confidential information.

well, this doesn't really help:

against E, who can read the password AND confidential info
against M, 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"

using?

in advance: share secret key

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

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

using?

in advance: share secret key

A computes E(key, ‘The secret formula is...’) $=$ ***

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

B computes D(key, ***) $=$ ‘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 $=$ [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" $\rightarrow$ "Pay \$999 to Bob"

we can shorten

"Pay \$100 to ABC Corp if they ..." $\rightarrow$ "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 $MAC(\text{key}, \text{other message})$

"forging 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

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

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

public keys

public key used to encrypt

can share this with everyone!

private key used to decrypt

kept secret

- don't even share with people sending us messages

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(private 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 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$\rightarrow$B: #50 Did you order lunch? [signature by M]
B$\rightarrow$M: #50 Yes. [signature intended for M by B]

A$\rightarrow$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 send C:

  "B's public key is XXX" AND
  Sign(B's private key, "B's public key is XXX")

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

instead, have public keys of trusted *certificate authorities*

only 10s of them, probably

websites go to certificates authorities with their public key

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

these signed messages called "certificates"

example web certificate (1)

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            81:13:c9:49:90:8c:81:bf:94:35:22:cf:e0:25:20:33
        Signature Algorithm: sha256WithRSAEncryption
        Issuer:
            commonName                = InCommon RSA Server CA
            organizationalUnitName    = InCommon
            organizationName          = Internet2
            localityName              = Ann Arbor
            stateOrProvinceName       = MI
            countryName               = US
        Validity
            Not Before: Feb 28 00:00:00 2022 GMT
            Not After : Feb 28 23:59:59 2023 GMT
        Subject:
            commonName                = collab.its.virginia.edu
            organizationalUnitName    = Information Technology and Communication
            organizationName          = University of Virginia
            stateOrProvinceName       = Virginia
            countryName               = US
.....
```

example web certificate (1)

```
Certificate:
    Data:
....
        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:
....
        X509v3 extensions:
....
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
....
            X509v3 Subject Alternative Name:
                DNS:collab.its.virginia.edu
                DNS:collab-prod.its.virginia.edu
                DNS:collab.itc.virginia.edu
    Signature Algorithm: sha256WithRSAEncryption
         39:70:70:77:2d:4d:0d:0a:6d:d5:d1:f5:0e:4c:e3:56:4e:31:
....
```

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)

backup slides