Nickolai Zeldovich

key ideas for today:
open design
identity vs authenticator
authenticating messages vs principals (message integrity, bind data)
public key authentication

--- new board ---

security model
C - I - S
last time:
talked about some general ideas for how to build secure systems
defensive design: expect compromise, break into parts, reduce privilege
last recitation, saw examples of what can cause parts to be compromised
for the rest of the lectures:
assume that we can design end-points to be correct & secure
(hard but let's go along with this for now)
figure out how to achieve security in the face of attackers
attacker can look at, modify, and send messages
basic goals that we want to achieve
inside the server: guard - service

authentication
authorization
confidentiality

--- new board ---

basic building block: crypto
let's look at how you might implement encryption
two functions, Encrypt and Decrypt
C -> E -> I -> D -> S

military systems, E and D are secret
closed design

problem: if someone steals your design, you're in big trouble
hard to analyze system without at the same time losing secrecy
key principle in building secure systems: minimize secrets!

--- new board ---

open design
big advantage: if someone steals design & key, can just change keys
can analyze system separately from the specific secret key
minimizes the secrets

important principle in designing systems:
figure out precisely what secrets distinguish bad guys from good guys
it's very hard to keep things secret
knowing what's important will allow you to focus on the right things

same diagram but with keys going into E & D
example of symmetric key crypto: one-time pad
XOR the message with random bits, which are the key
quickly describe XOR, why you get the original message back
problem: key is giant (but scheme is perfectly secure)

stream ciphers: various algorithms that generate random-looking bits
no longer perfectly unbreakable, just requires lots of computation
SLIDE: RC4

attack if keys reused
C->S: Encrypt(k, "Credit card NNN")
S->C: Encrypt(k, "Thank you, ...")

XOR two ciphertexts and known response to get unknown request message!
ever reuse keys with symmetric crypto! (one-time pad!)

--- new board ---

previously needed shared keys, doesn't scale

RSA: public-key cryptography
keys for encryption, decryption differ
SLIDE: RSA algorithm
short example computation?

\[ p = 31, \ q = 23, \ N = 713 \]
\[ e = 7, \ d = 283 \]
\[ m = 5 \]
\[ c = m^e \mod N = 5^7 \mod 713 = 408 \]
\[ m = c^d \mod N = 408^283 \mod 713 = 5 \]
difficult to generate e from d, and vice-versa
assumption: factoring N is hard!
much more computationally expensive than symmetric-key crypto!
important property: don't need a shared key between each party
encrypting a message for someone is different than decrypting it
server can use the same key for many clients sending to it

similarly tricky to use in practice
how to represent messages?
small messages are weak
large messages are inefficient
can multiply messages together
need something called padding

crypto mechanisms rely on computational complexity
pick key sizes appropriately -- "window of validity"

--- new board ---

principal authentication
principal/identity: a way of saying who you are
authenticator: something that convinces others of your identity
open design principle sort-of applies here
want to keep identity public, authenticator private
focus on what's distinguishing good guy from bad guy
usually there's a rendezvous to agree on an acceptable authenticator

authenticator types: right side of the board
real world: SSN
bad design: confuses principal's identity and authenticator
passwords
assuming user is the only one that knows password, can infer that
if someone knows the password, it must be the user
server stores list of passwords, which is a disaster if compromised
common solution: store hashes of passwords
   define a cryptographic hash:
      \( H(m) \rightarrow v \), \( v \) short (e.g. 256 bits)
      given \( H(m) \), hard to find \( m' \) such that \( H(m') = H(m) \)
      foils the timing attack we had last time
      in theory hard to reverse
   dictionary attack: try short character sequences, words
physical object
magnetic card: stores a long password, not very interesting
smartcard: computer that authenticates using crypto
biometric
oldest form of authentication: people remember faces, voices
can be easy to steal (you leave fingerprints, face images everywhere)
unlike a password, hard to change if compromised
more of an identity than authentication mechanism

need to trust/authenticate who you're providing your authenticator to!
fake login screen, fake ATM machine can get a user's password/PIN
next recitation you'll read more about what happens in the real world
web phishing attacks: convincing you to authenticate to them

--- new board ---

suppose we trust our client (e.g. laptop, smartcard, ...)
how to design protocol?
   board: C - I - S diagram
   client sending a message saying "buy 10 shares of Google stock"
   simple version: just send password over the network
   attacker has password, can now impersonate user
   better version? send a hash of a password
   attacker doesn't get our password (good, probably)
   but the hash is now just as good -- can splice it onto other msg!

** need both authentication AND integrity **

better? include checksum of message, eg CRC
   attacker can re-compute checksum! need checksum to be keyed

better yet: send a hash of \([ \text{ message + password } \] \), called a MAC
   message authentication code
   if you're going to do this: look up HMAC
   best: establish a session key, minimize use of password (long-term secret)
   send a message to the other party saying "i will use this key for a bit"
   use that key to MAC individual messages