Layer Optimization: Security and Privacy
CS 118
Computer Network Fundamentals
Peter Reiher

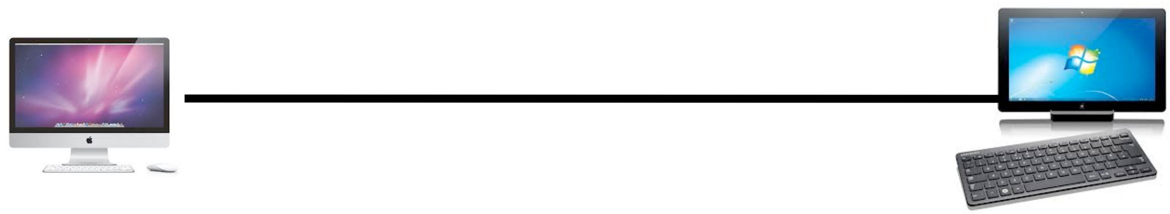
Another type of layer deficiency

- Some layers of protocol do not provide any security or privacy
- What if we want to have better security and privacy?
- What do we do to get them?

What do we mean by "security?"

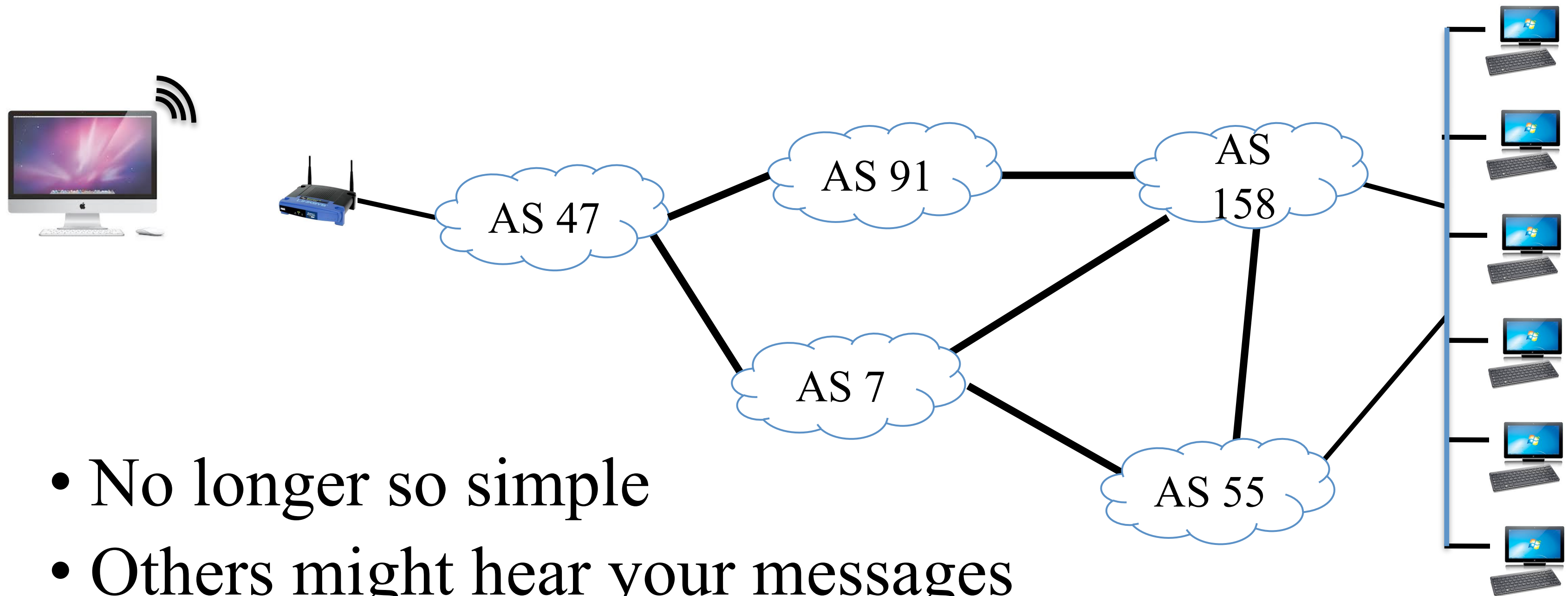
- Informally, providing some of three properties:
  - Confidentiality
  - Integrity
  - Availability
- In the face of adversaries attempting to compromise those properties

Security on a single link



- A relatively simple problem
  - If this picture is accurate
- Nobody else can hear your messages
  - So confidentiality is good
- Nobody else can alter your messages
  - So integrity is good
- Nobody else can interfere with your messages
  - So availability is good

Security in a complex network



- No longer so simple
- Others might hear your messages
  - So confidentiality is bad
- Others might alter your messages
  - So integrity is bad
- Others might interfere with your messages
  - So availability is bad

Authentication

- Proving that something was created by a particular party
- E.g., a message was created by the user who appears to have sent it
- Vital property to achieve many security goals
- Since sometimes you will do things for some parties, but not others
  - Only works out if you can tell who is who

Security

- Background
- Information protection
- Resource protection

Background

- Basic mechanisms
- Key management

Basic mechanisms

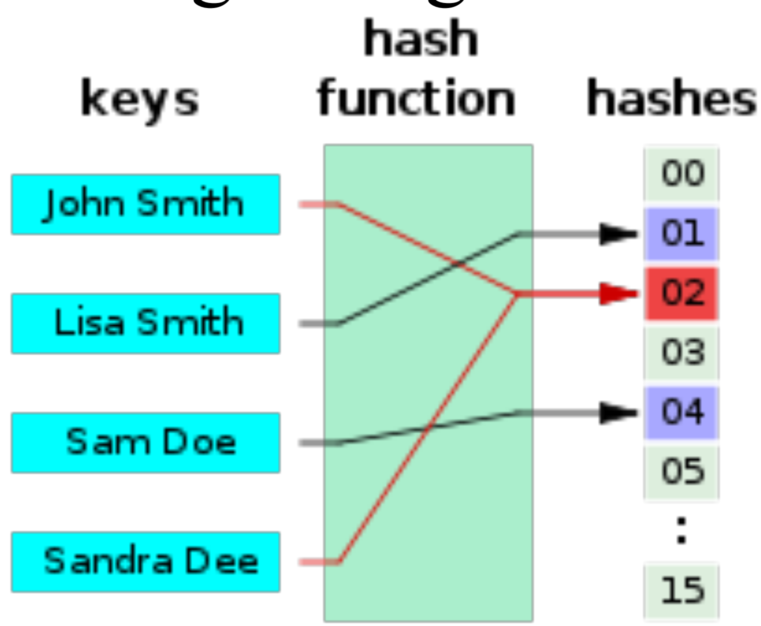
- For networks, primarily based on data manipulations
  - Hashes
  - Ciphers and codes
  - Signatures
- Also need to protect network resources
  - Need different mechanisms for that

Security by data manipulation

- Put (or alter) data in packets to improve security
- Hashing
  – Integrity (detect tampering)
- Encryption
  – Confidentiality (obscure semantics/meaning)
- Signature
  – Authentication (identify source)

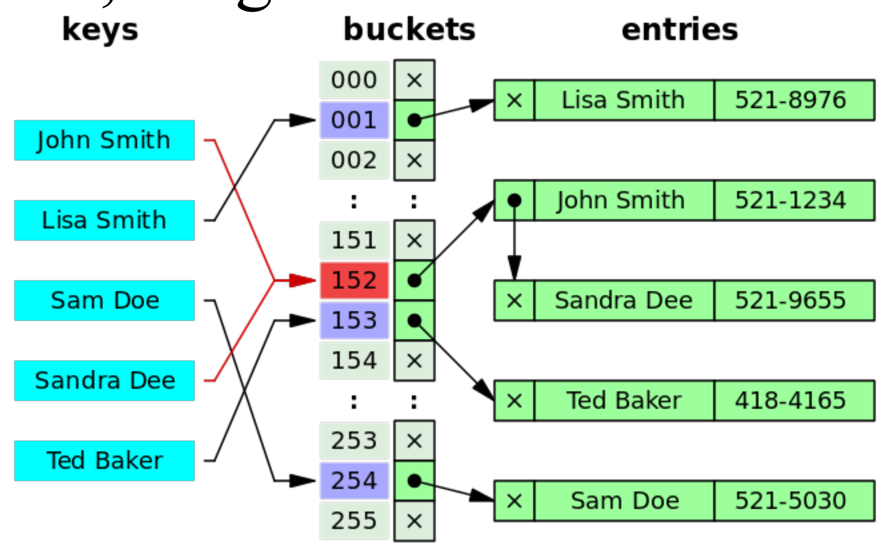
Hash functions

- Maps a variable-length message onto a fixed-length "digest"



Why hash?

- Scramble
  - Nearby messages yield very different digests
  - Distributes / scatters load, usage
  - E.g., hash tables
    - Rapid lookup
    - Avoids bunch-up
    - Collisions OK



Cryptographic hash

- A hash function useful for cryptography
  - Scrambles and spreads (like any hash)
  - Difficult to "game"
- Anti-"game" properties
  - Unidirectional (non-invertible)
    - Difficult to generate another input for a given output
  - Rare collisions
    - Difficult to generate two inputs with the same output

Why not just use a checksum?

- Checksums don't protect against tampering
  - Easy to generate a new message with the same checksum
  - Easy to generate two messages with the same checksum
- Cryptographic hashes do protect
  - Unidirectional and rare collisions make the above difficult

Example hash functions

- Message Digest 5 (MD5)
  - The 5th attempt at a message digest
    - MD, MD2, MD3, MD4, MD5, now MD6
  - Weak – found a birthday attack
- Secure Hash Algorithm (SHA)
  - SHA-1 is weak
  - SHA-2 and SHA-3 well regarded
  - US Government designed

What do you do with a hash?

- Publish it as-is
  - A “fingerprint” to validate as “untampered”
  - Assumes the published hash wasn’t tampered
- Use it in other algorithms
  - HMAC
  - Digital signatures

Fingerprint checks

- E.g., for GPG software

```
d065be185f5bac8ea07b210ab7756e79b83b63d4  gnupg-2.0.27.tar.bz2
091e69ec1ce3f0032e6b135e4da561e8d46d20a7  gnupg-2.1.3.tar.bz2
fb541b8685b78541c9b2fadb026787f535863b4a  gnupg-w32-2.1.1_20141216.exe
5503f7faa0a0e84450838706a67621546241ca50  gnupg-1.4.19.tar.bz2
d0cf40cc42ce057d7d747908ec21a973a423a508  gnupg-1.4.19.tar.gz
dc03ae4e4c3e8fe0583b37dd6c3124f94246d2f8  gnupg-w32cli-1.4.19.exe
4997951ab058788de48b989013668eb3df1e6939  libgpg-error-1.19.tar.bz2
9456e7b64db9df8360a1407a38c8c958da80bbf1  libgcrypt-1.6.3.tar.bz2
86fe0436f3c8c394d32e142ee410a9f9560173fb  libksba-1.3.3.tar.bz2
7cf0545955ce414044bb99b871d324753dd7b2e5  libassuan-2.2.0.tar.bz2
01e62c45435496ff0e011255fb0ac1879a3bc177  pinentry-0.9.1.tar.bz2
8dd7711a4de117994fe2d45879ef8a9900d50f6a  gpgme-1.5.3.tar.bz2
9eb07bcceeb986c7b6dbce8a18b82a2c344b50ce  gpa-0.9.7.tar.bz2
a7a7d1432db9edad2783ea1bce761a8106464165  dirmngr-1.1.0.tar.bz2
```

Any alternatives to hashing?

- Protect the path
  - Lock it down, seal it up, etc.
- Detect tampering
  - Power loss, other physical changes
- All are very hard to do

Encryption

- Convert an easily readable bit pattern into a bit pattern that looks very different
- Typically one that looks like random data
- Usually in a reversible way
  - So those you want to use the data can
  - Requires that not everyone can reverse it
- How to achieve that?

Keyed encryption

- Use a secret to perform the conversion
- If you know the secret, reversing it is easy
- If you don't know the secret, reversing it is hard
  - Preferably impossible
- The secret is called the *key*
- Leading to an obvious question:
  - How can I keep a secret by using another secret?

Symmetric and Asymmetric Encryption Systems

everyone knows my public key

encrypt with my private, and your public

decrypt with your private key and my public

- Symmetric systems use the same keys to encrypt and decrypt
  - Encrypt your data with key K
  - Decrypt and get the data back with K
- Asymmetric systems use different keys to encrypt and decrypt
  - Encrypt your data with K_E
  - Decrypt and get the data back with K_D
  - $K_E \mathrel{!}= K_D$

encrypt private, decrypt public to authenticate

authenticate my message - see who sent it

encrypt public, decrypt private to sent

closed book closed notes
no questions about 'do you know what this means..."
apply knowledge to practical problems
cumulative - write a couple of paragraphs - a page of text
no electronic devices in the test

concentration in the class - both readings and lectures
no need to memorize equations

Example codes

- Symmetric
  - Data Encryption Standard (DES)
  - Advanced Encryption Standard (AES)
- Asymmetric
  - Diffie-Hellman
    - They just won the Turing Award for inventing asymmetric crypto
  - RSA algorithm
  - Elliptic curve algorithms

Symmetric keys

- Also known as “shared secret”
  - Both sides share the same key
  - Both sides can encrypt or decrypt
- Generally faster than asymmetric crypto

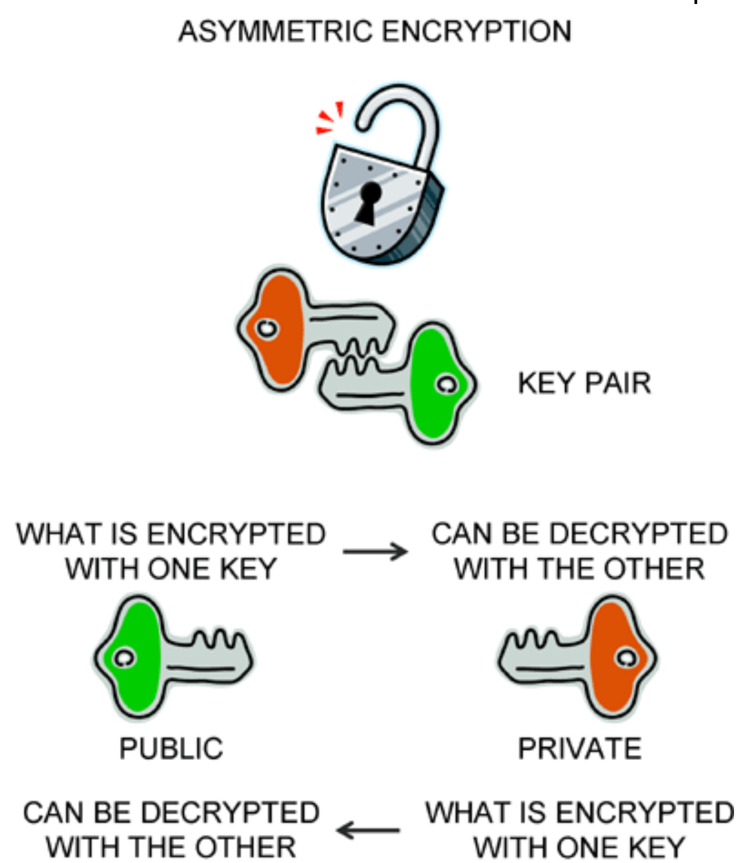
This turns out to be <u>really</u> important!

Using symmetric keys

- Assume the data you want to encrypt is P
- The encryption algorithm is E
- The decryption algorithm is D
- And the symmetric key is K
- $C = E(K, P)$
- $P = D(K, C)$
- Expanding, $P = D(K, E(K,P))$
- You end up with what you started with
- And you used the same key twice

Asymmetric keys

- *Public key* cryptography
  – Two keys: public and private
- To encrypt to a single recipient:
  – Anyone encrypts a message with your public key
  – Only you can decrypt with your private key
  – *Only you can read it*
- To identify a source:
  – You encrypt a message with your private key
  – Anyone can decrypt with your public key
  – *Only you could have written it*



Symmetric vs. asymmetric

- Symmetric: same key is used on both ends
  - Anyone who has the key can create the message
  - Anyone who has the key can read the message
  - Info is private to those who share the key
  - Info was created by someone who knew the key
- Asymmetric: keys are used as pairs
  - Public key creates message only private key can decrypt
    - Confidentiality – only private key owner can read it
  - Private key creates message only public key can decrypt
    - Authenticity – only private key owner could create it
    - But anyone can check ownership
- Again, symmetric is <u>much</u> cheaper than asymmetric

Using asymmetric keys

- Applying both keys yields the original message
  - $C = E(K_E,P)$
  - $P = D(K_D,C)$
- Or
  - $C = E(K_D,P)$
  - $P = D(K_E,C)$
- Unlike symmetric keys, the intermediates are different
  - $E(K_D,P) \mathrel{!=} E(K_E,P)$

Digital signatures

- Rely on asymmetric keys
  - Signer encrypts using their private key
- Entire message?
  - That's too costly
    - Remember asymmetric being expensive?
  - Less costly to sign a hash
- Signature
  - A cryptographic hash signed with a private key
  - A.k.a. Message Authentication Code (MAC)

Signatures and integrity

- Signature assures receiver of message integrity
  - Via the hash
- Contents haven't changed since the hash was computed
  - If they had, the hash wouldn't match
- Attacker can't just generate a new hash
  - Since it must be signed by the private key
  - Which he doesn't have
    - We hope . . .

Signatures and authenticity

- Signature assures receiver of authenticity
  - Message was created by the apparent sender
- Sender's private key was used to sign the hash
  - Hash came from the party with *that* private key
  - Which can only be the apparent sender
    - We hope . . .

Signatures and non-repudiation

- Signature prevents sender repudiation
  - Sender can’t deny it sent that message
- Why not?
  - The decrypted signature matches the message hash
  - But it’s a cryptographic hash
    - So it’s not likely the message could be changed to match the signature
    - OR that a signature can be reused for a different message

What do we have so far?

- Hash
  - Integrity, if you trust the hash
- Encryption
  - Privacy, given a key
- Signature
  - Authentication and integrity, given a key

"given a key"

- Security's three most feared words
  - Keys need to be shared in advance
    - Both sides have the symmetric key
    - Both sides have part of an asymmetric key
- The two challenges:
  - Endpoints need to know which key to use
    - Using the wrong key ruins everything
  - Endpoints need to get (and trust) the key
    - And if anyone else gets a symmetric key, you're screwed

Key management

- Pre-shared
- PKI
- Key exchange
- Keyless

Pre-shared

- Both sides share the key in advance
  - Technically, this is usually assumed
  - Typically referred to as "out-of-band" distribution
    - Out-of-band = someone else's job
    - I.e., "I'm not solving the hard part of the problem"
  - Useful for any keys
    - Shared secret (symmetric)
    - Public key (asymmetric)

PKI

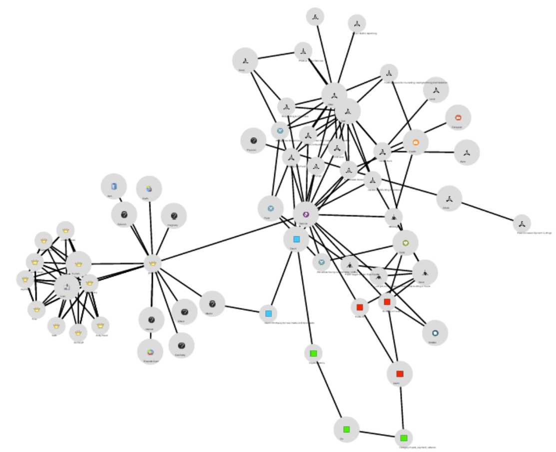
- Public Key Infrastructure
  - Using a database to get a key
  - Same as most other network databases
    - Distributed vs. central
    - Flat vs. hierarchical
    - Structured vs. "hash tree" (destroys locality)
  - Infrastructure for public keys
    - Useful only for public part of asymmetric keys
    - Not a "public infrastructure for keys"

PKI example

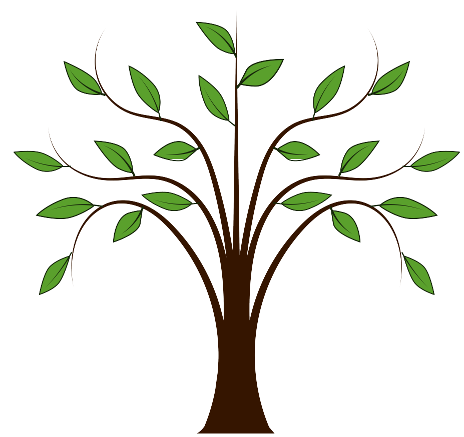
- PGP keys (e-mail)
  - Set of servers hold public keys
  - Users find keys explicitly
- X.509 keys
  - Key signed by a hierarchy
  - Many roots (built-in to browsers)
  - Can add others (self-signed, other-signed)

PGP vs. X.509

- PGP
  – Web of trust
  – Users sign each other's keys
  – Key signing "parties"
  – Trust based on who YOU trust



- X.509
  – Hierarchy of trust
  – Roots sign keys
  – Companies charge to sign keys
  – Trust based on "anchors" (roots)



Key Exchange

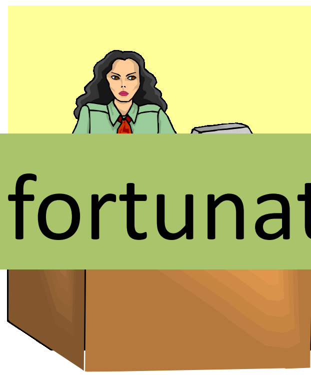
- Let’s say you want to communicate
- But you don’t share a key
  - Or you want to use a new key
  - Generally good not to use a single key too much
- Then you need to exchange a key between the communicating partners
- How?

Key exchange using both symmetric and symmetric crypto

- Common to use both in a single session
- Asymmetric cryptography essentially used to "bootstrap" symmetric crypto
- Use RSA (or another PK algorithm) to authenticate and establish a *session key*
- Use AES with that session key for the rest of the transmission

Combining Symmetric and Asymmetric Crypto

Alice wants to share the key only with Bob



Unfortunately, it's more complex than this

Only Bob can decrypt it

Alice

Bob

K_{EA} K_{DA}

K_{EB}

Only Alice could have c ted it

K_{EB} K_{DB}

K_{EA}

Take CS 136 if you'd like to know why

$C=E(K_S,K_{EB})$

K_S

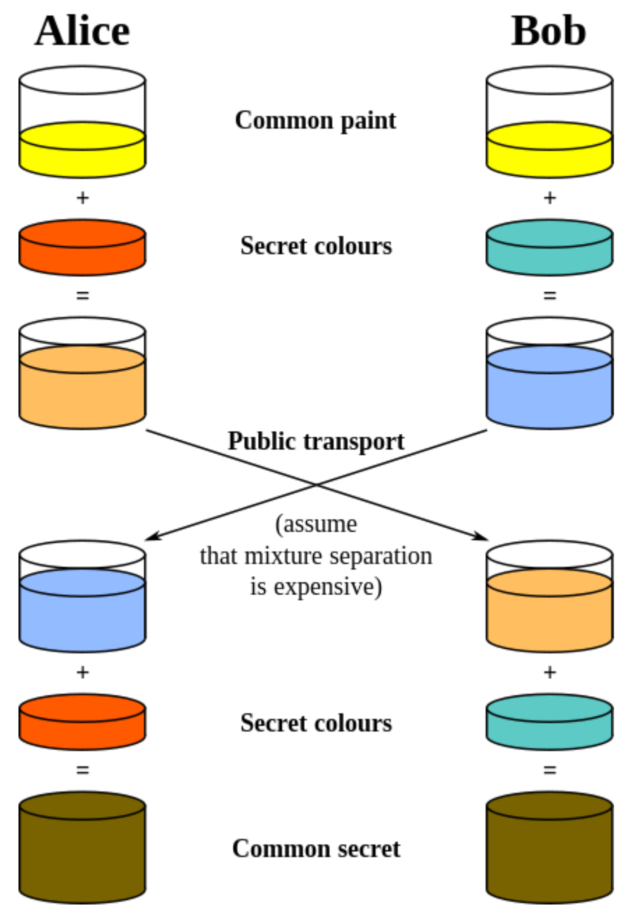
$M=E(C,K_{DA})$

$=D(C,K_{DB})$

$C=D(M,K_{EA})$

Diffie-Hellman key exchange

- Share a key starting from nothing
- Diffie-Hellman
  - Establish a shared secret over a public net
  - Each side has a secret (pick a random number)
  - Both sides share a common value
  - Relies on non-inverting mixing



How to mix irreversibly?

- Multiply
  - A x B = C
  - Assumes A and B are prime
  - Factorization is hard
- Exponentiation
  - $(a^x)^y mod\ p = (a^y)^x\ mod\ p$
  - Discrete logarithms are hard

What does DH do?

- DH establishes a shared secret
  - A symmetric key
- But symmetric keys don't establish identity
  - "Man in the middle" attack
  - Who do you share the secret with?
  - Solution: signed DH

Signed DH

- Use public key cryptography
  - Sign the messages (encrypt with private key)
  - Prevents MITM attack
- But if we have public key, why use not use that?
  - Remember how important it was that asymmetric crypto was expensive?
- Why not what we did a few slides ago?
  - Requires fewer of those expensive PK operations

Keyless

- The I in PKI is a pain
  - The hardest part of PKI is the key database
  - Everyone has to have a key
  - Everyone has to find the other party's key

- Solutions
  - Make the "I" easier (automate, etc.)
  - Avoid the "PK"

Keyless keying

- What if we use DH without signatures?
  - I.e., original DH

  - Share a secret – but with whom?

- What if we don't care?
  - WHO isn't know
  - But the rest of the exchange is protected

- "Better than nothing security" (BTNS)
  - Protects against others interfering

BTNS protection

- The connection is secure!
  - But to whom?
    - Who cares!
    - Once you start a conversation, you can’t be interrupted
    - Maybe you can somehow verify identity later?

Are we done?

- What have we protected?
  - Integrity
  - Privacy
  - Origin

- What have we not protected?
  - Resources!

Resource protection

- Endpoint
- Forwarding

Endpoint resource protection

- Resources to protect
  - Buffers memory
  - Processing / CPU processing
  - Content – the FSMs
- Ways to protect the endpoint
  - Shed load
  - Verify before acting

hard to figure out everything you should drop

drop traffic - drop al the junk; don't drop a single packet of what you don't want

Typical endpoint protections

firewall - sits at a place where it checks packets
get rid of traffic that you don't want your server to be handeling
examine the incoming traffic - see which one you like

- Rate limiting
  - Limit investment in new connections
  - Toss out when beyond a limit
  - Protect against SYN flooding attacks
- Firewalls, port blocking
  - Fixed: drop all packets to a particular port and/or in a particular direction
  - Conditional: drop a port until you know better

SYN flood - TCP SYNs are sent, but never finishes the connection
limit your resources, you don't want incoming traffic to set up resource for connection

Conditional port blocking

- NATs
  - Network and port address translator
  - Private and public side
  - Fixed: public side -> private side
  - Conditional: private side -> public side

- Conditional example
  - Allow incoming only if outgoing
    - Wait for DNS UDP out, allow response back in
    - Wait for TCP SYN (open), allow replies until FIN (close)

encrypt internal packet - wrap with another IP header - this is from IP address in starbucks
gets delivered to UCLA tunneling layer - decrypts and get IP packet

Variable load shedding

load shedding - too much work, time to drop something

- Port blocking: drop based on partial work
  - Examine addresses, ports, some content
  - Drop before investing more work
- Cipher/code/sign: drop on separate work
  - Validate (decrypt, authenticate) based on an algorithm that is separate from the FSM of the protocol
  - Drop before performing separate work
- Both attempt to separate security from FSM
  - Checking is distinct from acting on the message

exceptiosn to encrypt header

link - important
tunneling - have protocol
hide the protocol we want to
run, throw another layer
on top of things

tunnnel IP on top of HTTP

encapsulate IP on top of
HTTP

VPN - tunneling

don't encrypt the IP header on packets -

tunneling always has some sort of cost

runnel any protocol on top of another protocl - tunnel UDP on top of TCP

Forwarding resource protection

- Routers have two distinct roles
  - Relaying messages
  - As endpoints of routing and control protocols
- Two kinds of protection
  - Endpoint-like
  - Forwarding focused

Router endpoint protection

- Similar to other endpoints
  - Block ports
  - Limit rate
  - Validate content
- But a little harder sometimes
  - Bellman-Ford relays content indirectly
    - How can you protect the FSM?
    - Do you attach signatures for all the path components?
    - How does this affect scalability?

Forwarding protection

- Why would I refuse to forward a packet?
  - Others similar packets are causing a problem here
    - Overloading of security processing
    - Overloading my buffers
  - Other similar packets are causing a problem elsewhere (most commonly downstream)
    - The other end of the link has no room for it
    - The other end of the link says these are a problem
  - It never should have come to me
    - "Reverse path" checks

DDoS attacks - call sys admin
analyze your traffic

defense for DDoS attacks

Where do we put security?

- Everywhere you want to protect resources
  - Everywhere in the DAG
  - Sometimes at every hop, sometimes on ends
- In layers
  - Always better to shed load earlier in traversal
  - Don't always have enough info until later

Summary

- Security happens everywhere
  - At every layer
  - As early as possible, but no earlier
- Security uses several tools
  - Most based on math
  - Not all based on pure math – lots of alchemy
- Security assumes shared secrets
  - Like naming, it can't start from nothing