PKI (Public Key Infrastructure)

- Certificates + Certificate Authorities (CA):
  - Certificate: public key + "identifier"/name, bound together and signed by a trusted party

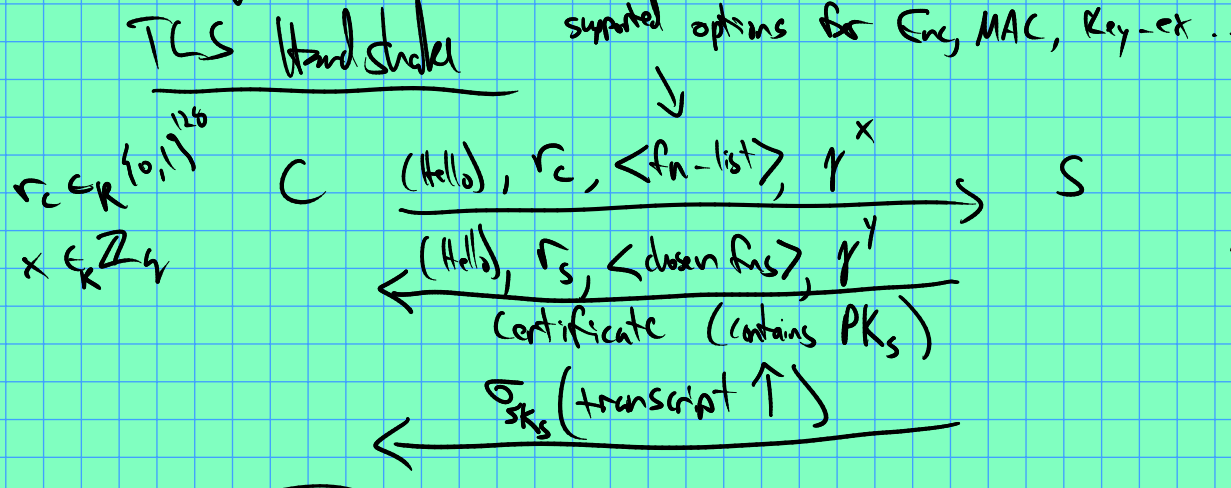
- Idea: "bootstrapping" trust: from a list of trusted parties (CAs) we can communicate securely (w/ authentication) with other parties to whom we've never before spoken

https: http over TLS/SSL

TLS sessions

goals: authenticate server to client, establish keys for encryption/MAC.
(Each message after the session is established will be authenticated + encrypted)

(1.3) TLS Handshake



Now C, S can compute keys...

(Separate keys for Enc, MAC, separate IV's

$k_{enc,s} = KDF(g^{xy} \| \text{"key for MAC"} \| \text{key length} \ldots)$)

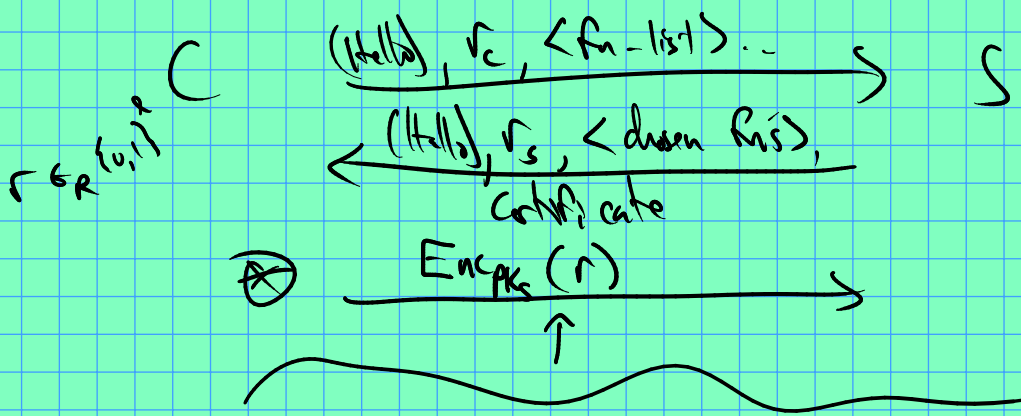
At this point, all further messages are
① Encrypted, ② MAC'd, ③ have a "seq. num." attached.

(Finished) MAC(transcript) ←

(Finished) MAC(transcript) →

Note: sequence no./nonces are incremented w/ each message.
Note: ↗ prevents "replay" attacks!

Note: older versions did things a bit differently:



Keys obtained via $KDF(r_c \| r_s \| r)$ ...

Issue: not forward Secure!

If SK_s is compromised, all past messages are decodable!!

New TLS (1.3) requires solving a new hard problem for each session. (E.g. fresh D-H instance.)

More advantages: protection against timing attacks.
(of new TLS)

Recall how RSA decryption of c could be sped up by converting to CRT form first:

$$c \underset{CRT}{\equiv} (c_p = c \bmod p,\ c_q = c \bmod q)$$

Then compute $\left(c_p^{d \bmod (p-1)},\ c_q^{d \bmod (q-1)}\right)$

and apply the inverse CRT.

If a server does this for (old) TLS handshakes, one could use timing info. to learn p, q! $(n = pq)$

Idea: in the message $C \xrightarrow{E_{PK_S}(r)} S$,
client chooses r. Start by choosing $r < p, q$,
and measure the timing of server's response.

Observation: if $r < p, q$, computing $r \bmod p$, $r \bmod q$
is trivial.

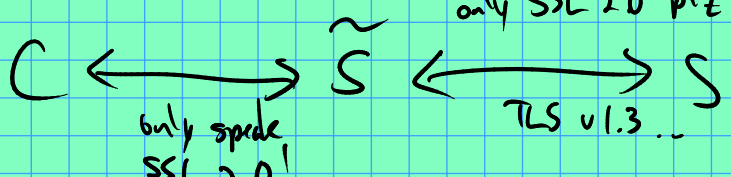
Now choose large $r > p, q$ and measure timing.

Now... binary search!

For a real world example, see Boneh et al ($\approx$ 2003?)
"Remote timing attacks..."

Other security concerns for TLS:

- Downgrade attacks



Mitigation in TLS 1.3: if downgrading, set last N bytes of r_S to a special string "I AM Downgrading!!"

if client did not request a downgrade and sees r_S ends in this string, end connection!

Certificate Revocation

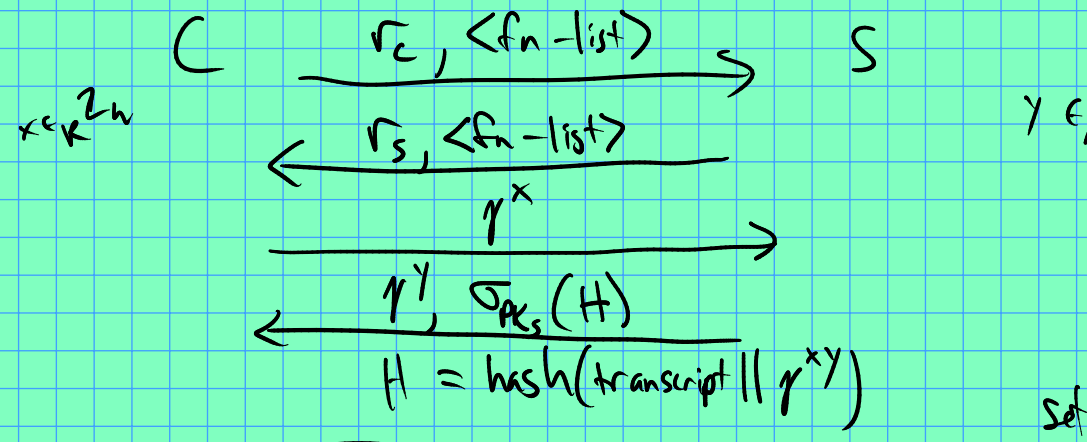
CA's maintain lists of revoked certificates. Should be checked... but... now a third party must be online! Not to mention privacy...

"microsoft.com\0\0\0"

Not addressed by TLS: privacy/anonymity

SSH (Secure Shell)

Handshake / session key establishment:



Now compute keys:

$$K_{enc,c} = KDF(K \| H \| \text{"A"} \| session\text{-}id = H)$$

$K_{enc,s}$ = " "B" ...."

Note: PK_s usually recvd by client directly (usually SSH does not use PKI).

See ~/.ssh/known_hosts for the list of servers you "know".

Note! order of MAC/Encryption can matter!

$Enc(m \| MAC(m))$ vs $Enc(m) \| MAC(c)$? (c = $Enc(m)$)

① ②

① is not "generically" secure, and in fact has been exploited! TLS used option ①, but SSH uses ②.

How to authenticate users?

Protocol allows several methods: passwords, "host-based", public key.

Remember best practices! → $SHA256^{(10000)}(pwd \| salt_{user})$

My favorite: public key!

(Note: common location for user keys: ~/.ssh/authorized_keys)

C → S: PK_c, username

C ← S: OK, PK_s

C → S: σ_{sk_c}(session-id || username || ... || PK_c)

Note: Above is all encrypted, so username hidden from adversaries.

Forward secrecy? ✓ (if using D-H)

Note: authentication is not deniable. (As you would want in messaging apps / OTR)

(Forward Secrecy: compromise of a long term secret does not affect secrecy of past sessions.)

Good: D-H each time

Bad: C $\xrightarrow{E_{PKS}(r)}$ S , K = KDF(r) ..

Cool SSH Tricks:

— SOCKS Proxy (tunnel your browser traffic)

ssh -ND <port#> <servername>

local port!! 12345

Then look under the network settings in d.f. and type in localhost as proxy w/ port 12345 !

— transfer files!

ssh server 'tar -C /tmp -cf music' | \

ssh raspberry 'tar -C /var/music/ -xvf -'

Spooky stuff: timing attacks.

usually if running a shell, each character typed sends a packet!

Can detect when a password is being typed...

and can measure the timing between key strokes

Might reduce pwd entropy considerably!!

(see D. Song ≈ 200X)