# last time

# 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)

# exercise

exercise: how should 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 identity validation

options involve CA choosing random value and:

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

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

exercise: problems this doesn't deal with?

# motivation: summary for signature

mentioned that asymmetric encryption has size limit

same problem for digital signatures

solution: sign "summary" of message

how to get summary?

hash function, but…

# 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) $=$ H

# cryptographic hash uses

find shorter 'summary' to substitute for data

- what hashtables use them for, but...
- we care that adversaries can't cause collisions!

# cryptographic hash uses

find shorter 'summary' to substitute for data

- what hashtables use them for, but...
- we care that adversaries can't cause collisions!

deal with message limits in signatures/etc.

password hashing — but be careful! [next slide]

constructing message authentication codes

- hash message + secret info (+ some other details)

# password hashing

cryptographic hash functions are good at requiring guesses to ‘reverse’

problem: guessing passwords is very fast

solution: slow/resource-intensive cryptographic hash functions

- Argon2i
- scrypt
- PBKDF2

# 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

alternate model:

- use public-key encryption like 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 B to get number

and b/c of math chosen, both get same number

# Diffie-Hellman key agreement (1)

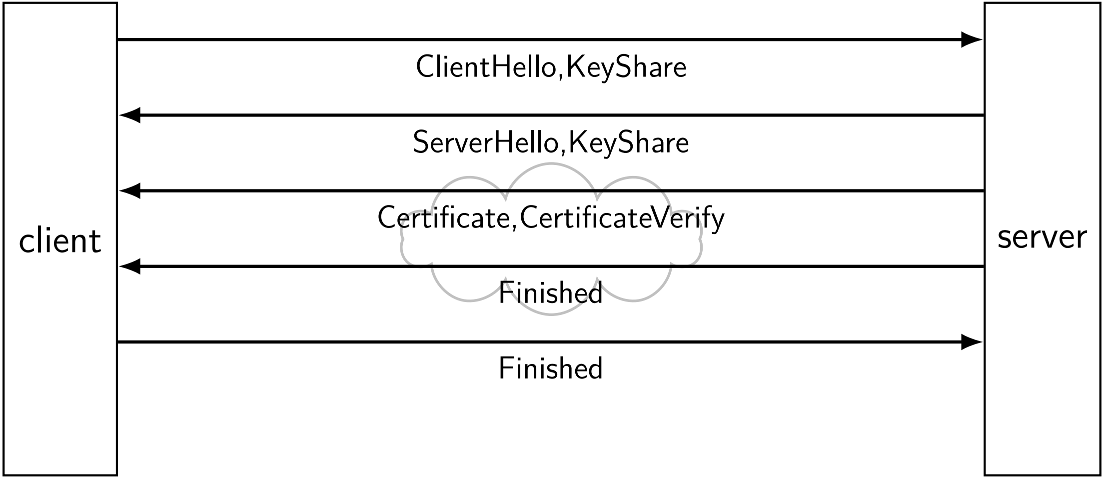
math requirement:

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

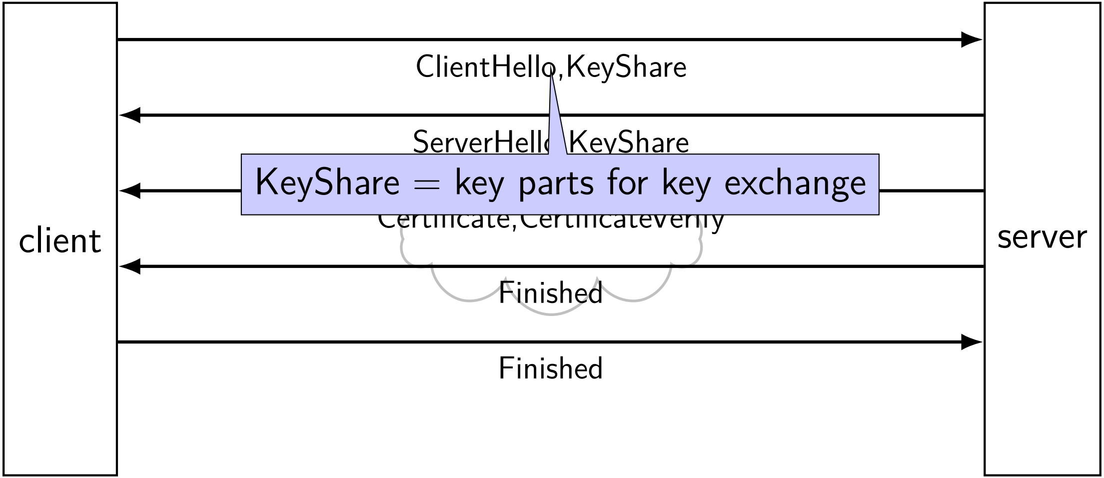
choose X in advance and:

| | |
|---|---|
| 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 $f(f(X,Y),Z)$ |

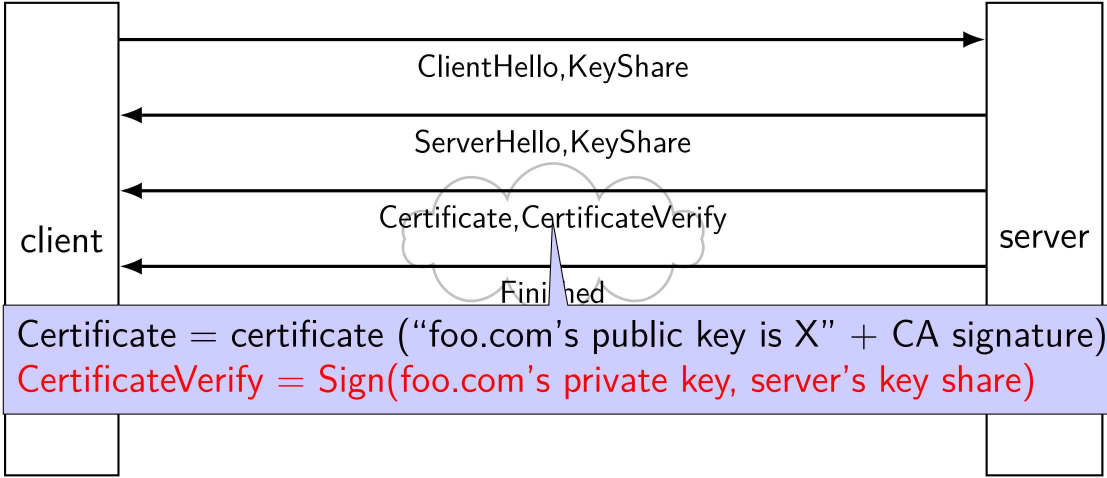
# typical TLS handshake



# typical TLS handshake

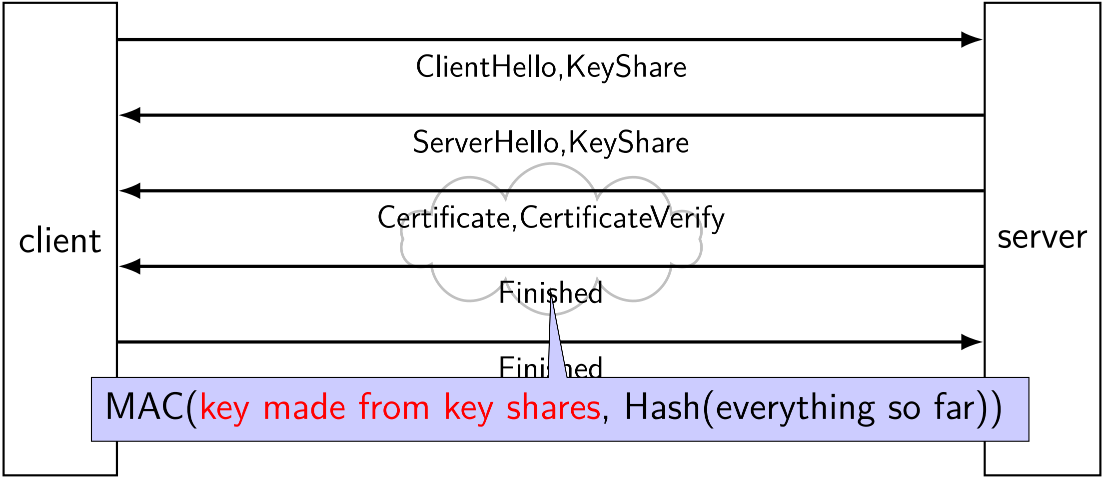


# typical TLS handshake

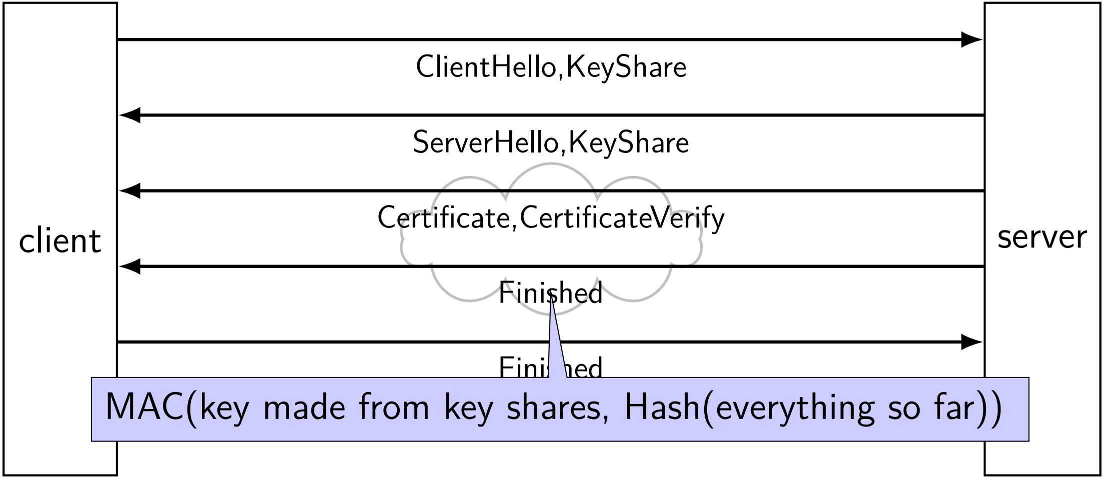



Certificate = certificate (“foo.com’s public key is X” + CA signature)
CertificateVerify = Sign(foo.com’s private key, server’s key share)

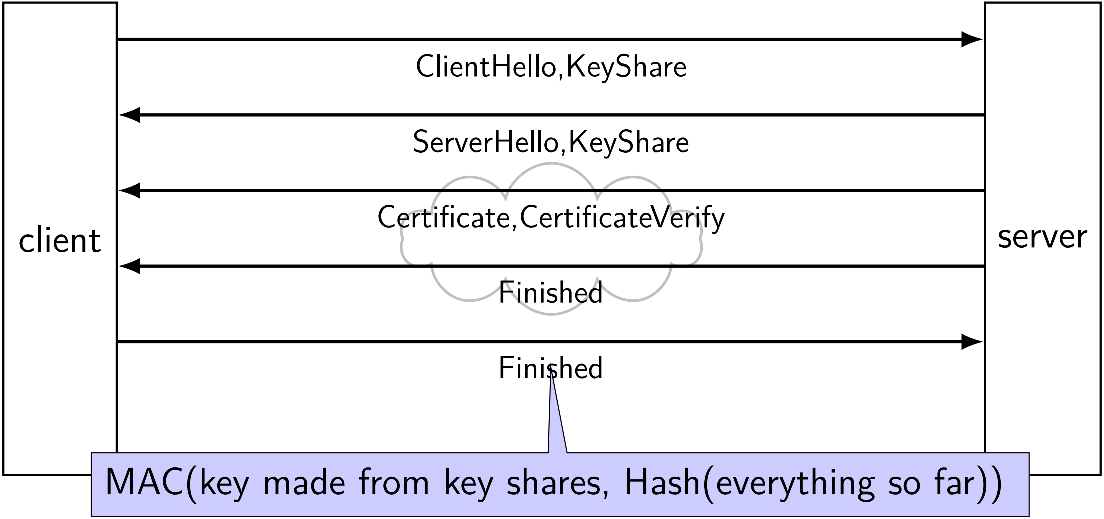
# typical TLS handshake



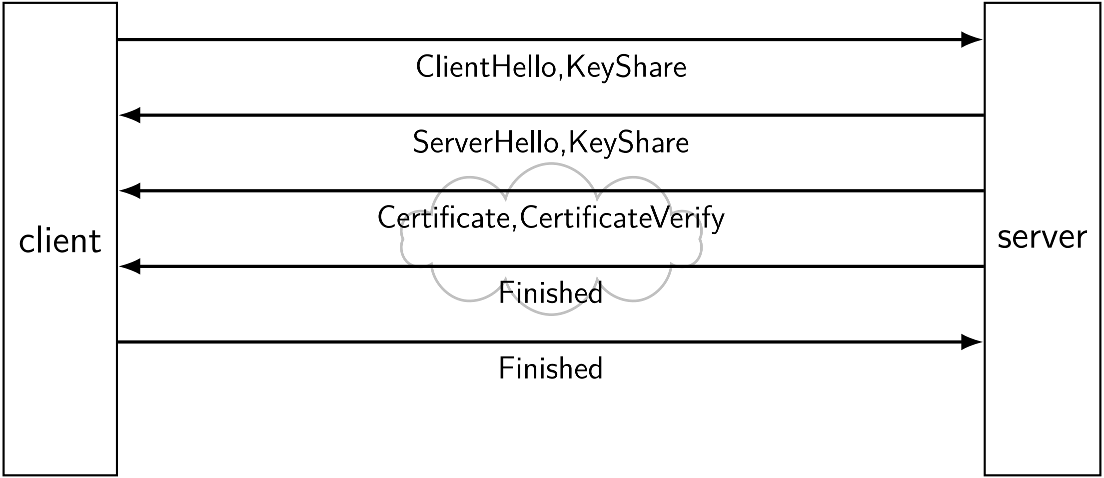
# typical TLS handshake



# typical TLS 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 → client and vice-versa)

often separate keys for encryption and MAC

later messages use encryption + MAC + nonces

# denial of service

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

...

# backup slides