transport/app-layer: ssh & ssl

Network Mgmt/Sec.

Outline

- secure shell
  - intro/how to use it/features, etc
  - v1 protocol
  - v2 quick overview
- secure socket layer (ssl)
  - intro
  - protocol
- conclusion - what are trust models?

what is secure shell?

- a secure replacement for rlogin/rsh/rcp OR telnet/ftp
- slogin/ssh/scp clients talk to sshd, tcp, port 22
- in two versions ssh 1.0 and ssh 2.0
- 2.0 is undergoing IETF standardization (secsh WG)
  - differences are incompatible and involve precise details of security protocol, and key mgmt. capabilities

just to be clear

- for **rlogin**, we have **slogin**
- for **rsh**, we have **ssh**
- for **rcp**, we have **scp**
- for **rshd**, we have **sshd**
  - /usr/local/sbin/sshd (sshd1 and sshd2 from v2 POV)
- and less reason to use telnet and ftp with a password

features

- public-key based authentication
  - in v2, may be able to use various kinds of certificates as well
  - in v1, RSA keys, and/or better password authentication
    - » password NOT sent in clear, but encrypted with server-side public-key
- bulk packets are encrypted (+ MAC used)
- /etc/hosts.equiv and ~/.rhost NOT used

features, cont.

- support exists in addition for port redirection “tunnels” or connections between ssh capable hosts
- also can automatically run remote X-clients over ssh connection to local X-server
  - thus make X more secure
- or simply redirect other insecure TCP services to use ssh (e.g., email)

Secure Shell - bibliography

- ssh, v1 IETF-draft, draft-ylonen-ssh-protocol-00.txt, "*The SSH Remote Login Protocol*", T. Ylonen, Helsinki, 1995
- IETF secsh working group, drafts include:
  – protocol architecture
  – transport protocol
  – authentication sub-protocol
  – connection sub-protocol

URLs of interest

- www.ssh.fi - home
  - mirrors exist (can download free unix, purchase windows slogin)
- www.ietf.org/html.charters/secsh-charter.html - IETF wg
- www.tac.nyk.ny.us/~kim/ssh Getting Started With SSH (tutorial)

v1 authentication mechanisms

- can include rsh compatibility with /etc/hosts.equiv, and .rhost, bu t why?
- password authentication
  - password is encrypted with server-side RSA public-key, NOT SENT IN CLEAR
- RSA-based authentication using ssh-keygen and passphrase

v1 encryption can include

- idea - 128bit key (default)
- des, 3des
- blowfish
- rc4

how to install v1

- ftp a copy ... (say 1.2.27) and tar -xzvf ssh*
- in build directory
  - ./configure
  - make install
  - add /usr/local/sbin/sshd to /etc/rc.local
    - » or /etc/rc.d/rc.local :->
  - run /usr/local/sbin/sshd &
  - look over config file in /etc

strong piece of my mind ...

- turn off rsh and friends
- /etc/inetd.conf, comment out and restart inetd
  - # shell ... rshd
  - # login ... rlogind
  - kill -HUP `cat /var/run/inetd.pid`

how to use it

- assume jrb is here and wants to login as jrb "there"

  % slogin [-l jrb] there.cs.pdx.edu

  – prompted for password or passphrase
- assume I want to run a command remotely

  % ssh there.cs.pdx.edu ls -l

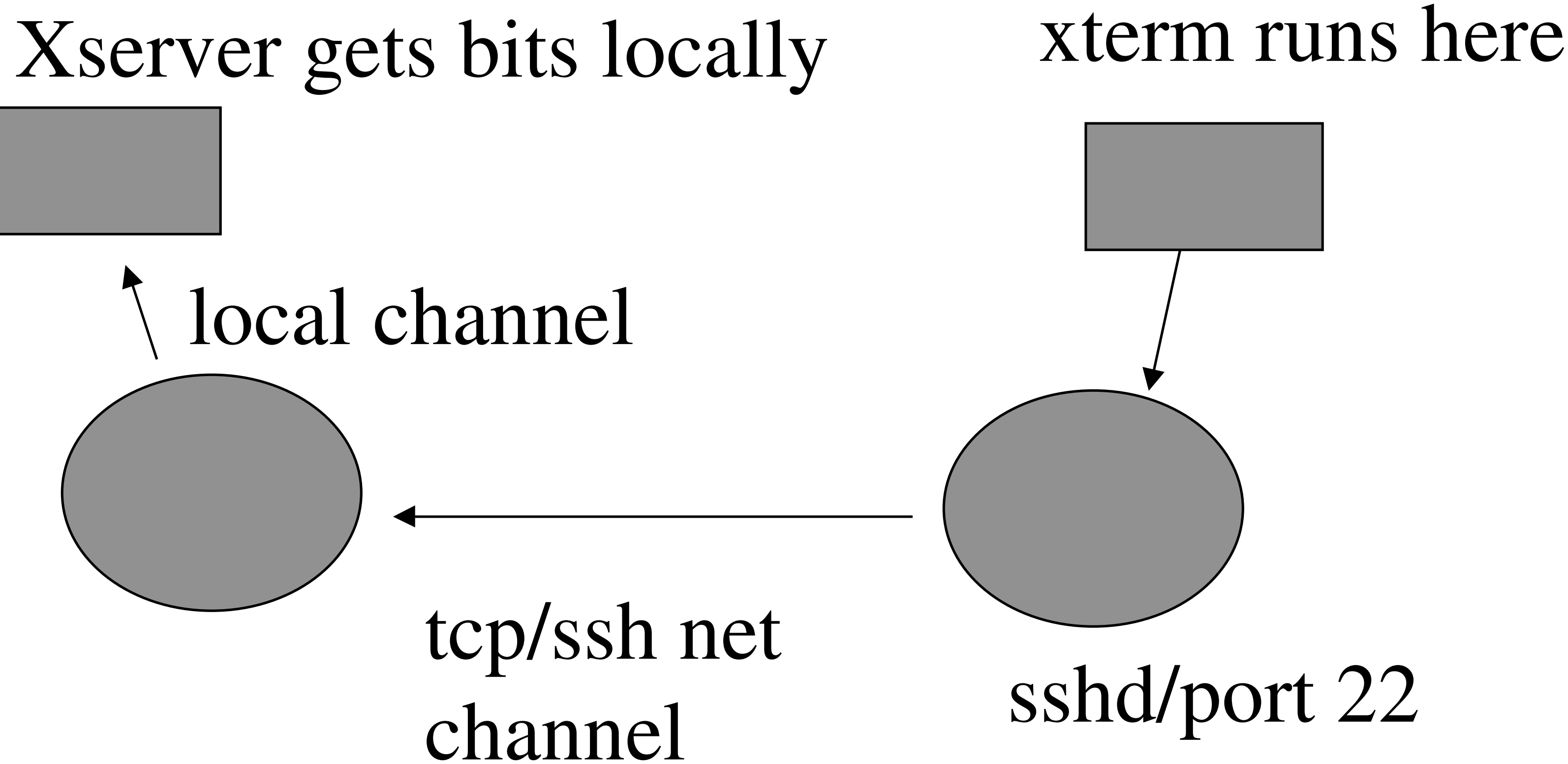
file copy (unix)

- from here to there
  - **% scp here.txt jrb@there.cs.pdx.edu:**
  - file ends up as here.txt on ~jrb on there
  - you can of course do absolute copy
- from there to here
  - **% scp jrb@there.cs.pdx.edu.:here.txt here2.txt**
- recursive directory copy
  - **% scp -r here.dir jrb@there.cs.pdx.edu:**

the remote X trick

- you are on here, just slogin to there
- **(there)% xterm &**
- it should show up “here” ...
- you don’t need to muck around with your environment over there, and do xhost “here”
- **ssh -n there.cs.pdx.edu xterm &**
  - in one swell foop ...

X11 connection forwarding



ssh-keygen in v1

- to create RSA key on here for there
- % ssh-keygen
  - in ~user/.ssh we have
    - » identity (your private key)
    - » identity.pub (your public key)
    - » authorized_keys (key collection from other systems)
- take identity.pub (ASCII) and move to there
- concatenate at end of .ssh/authorized_keys
- now use passphrase to "login"

ssh-agent

- can be run behind the scenes to hold keys for remote systems
  - and minimize use of passphrases (just once)
- should consider carefully exposing private key on multi-user system ... (may not want to use this)

v1 protocol description

1. client (C) TCP connection to server (S)
2. S sends ASCII version ident string
3. C sends own version in ASCII

both sides learn about version and can make compatibility decisions

4. both sides switch to binary packet-based mode
5. server sends RSA public-key (host key) + regenerated RSA key on per hour basis

protocol description, cont.

6. client generates 256 bit session key, encrypts with RSA keys, sends encrypted session key to S. list of algorithms for bulk payload included.
7. both sides then begin to encrypt
8. client may choose to request allocation of ptty, start X11, or do tcp/ip port forwarding, auth. agent forwarding, shell or command execution

authentication with password

- authentication sub-phase must occur before remote shell executed
- distinct authentication phase POST session key/crypto setup
- client sends server user-name
- client sucks in password from user post password prompt
- encrypts and sends to server
- server either accepts or fails ...

authentication with user-side public-key

- client has a priori stored public key on server
- server creates challenge with client public key, sends to client
- client must decrypt with private key, and other crypto dressing
- returns MD5 hash to server under startup encryption mechanisms

note one security tradeoff

- client has to believe server public key a-priori
  - otherwise possible man-in-the-middle attack
  - client-side can store public keys and make sure they don't change
  - or in v2, begin to use a priori server certificates or somehow (TBD) access Public-Key Infrastructure

v2 differences (quick/dirty)

- required public-key algorithm is DSS [FIPS-186], Digital Signature Standard, 1994
- v1 trust model is that client has local key database of public keys
  - manual ...
- v2 adds possibility of storage of single CA for multiple keys (of course ... multi CAs likely ...)
- formalizes transport with sub-protocols for authentication and connection (port-forwarding)

key exchange

- 1st do algorithm negotiation
  - (cookie, algorithms including encryption/mac/compression/languages)
- authenticated key exchange follows
- rekeying may occur from either side
  - on time or data basis (one hour or 1 gbyte)

v2, cont.

- explicit use of Diffie-Hellman to negotiate session keys
- MAC algorithms used include: hmac-sha1 (required), hmac-sha-96, hmac-md5, hmac-md5-96
- public-key, certificate formats include:
  - ssh-dss (required)
  - x509v3 (recommended), SPKI, OpenPGP

ssl - secure socket layer

- intro
- the protocol
- a bit on servers/certificates

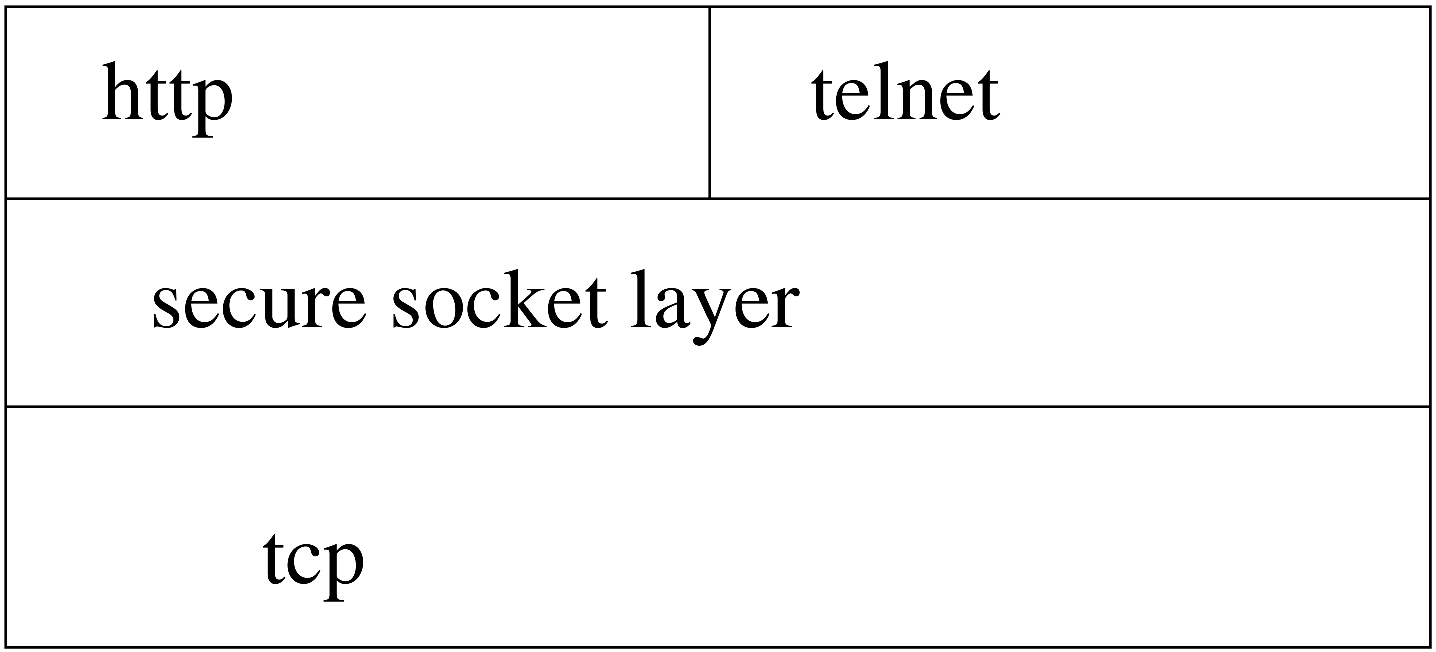
ssl - secure socket layer

- originally developed by netscape for web client to web browser security
- SSL designed to be under HTTP
  - HTTP | SSL | TCP
  - unlike SHTTP which is security IN HTTP
- can in theory be used with non-HTTP based protocols
  - experiments exist; e.g., telnet over SSL

bibliography

- William Stallings, Cryptography and Network Security, 2nd edition, 1998
- SSL at netscape, including v2/v3 (e.g., home.netscape.com search on certificate or ssl)
- RFC2246 - The TLS Protocol, Version 1.0, Dierks, Allen, Jan 1999
  - it's SSL v 3.1
- www.openssl.org - SSLeay library
- www.apache-ssl.org (also www.apache.org) - apache with ssl

ssl layering



very session-layer hmmm...

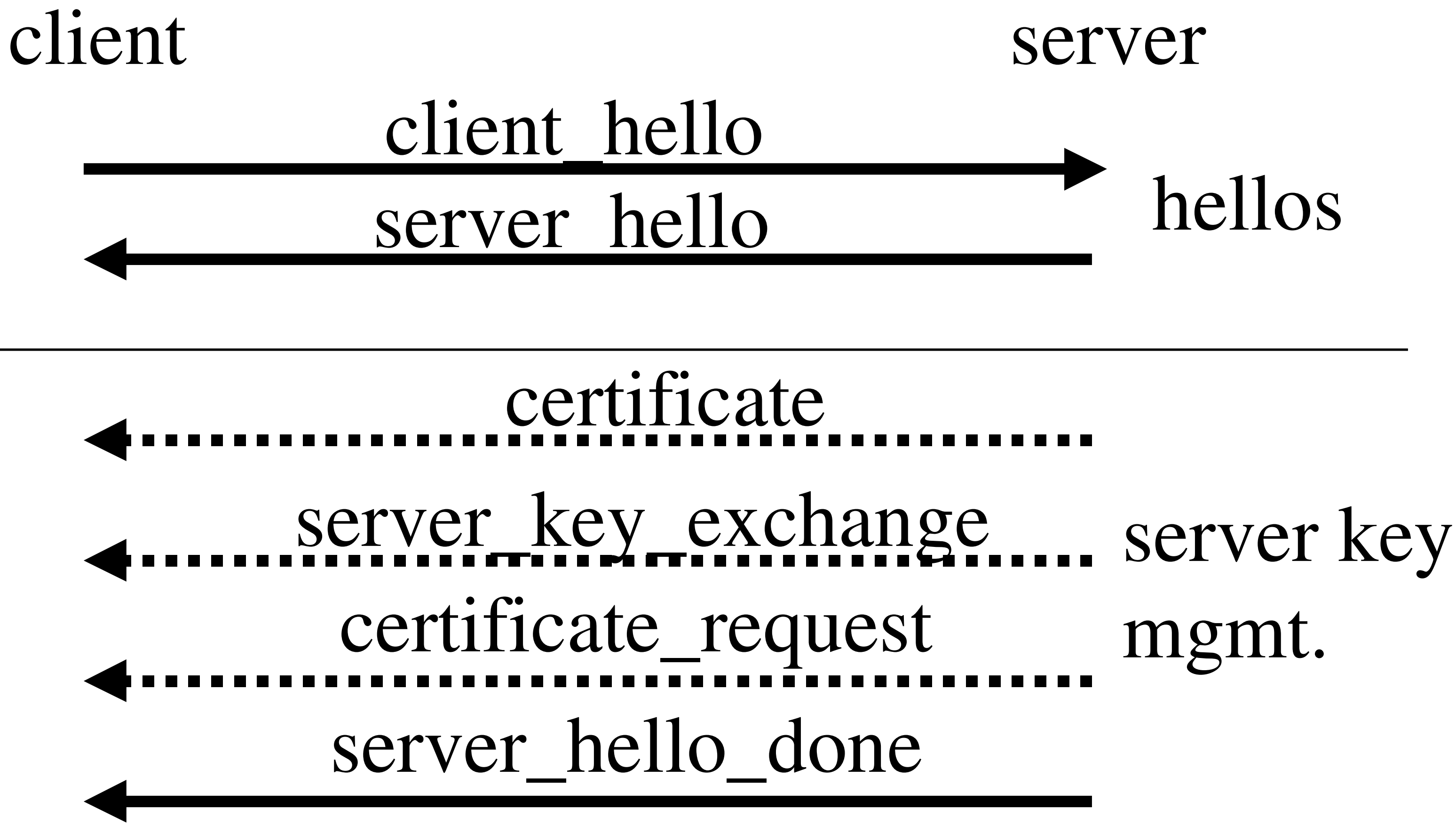
sub-protocol/s in SSLs

- ssl record protocol - bulk crypto; i.e., the result of initial negotiation and per packet
  - telnet/http packets encapsulated
- ssl handshake protocol - initial startup, session-key/crypto transforms decided
- change cipher spec - causes pending cipher state to become real state (trivial protocol)
- alert protocol - errors

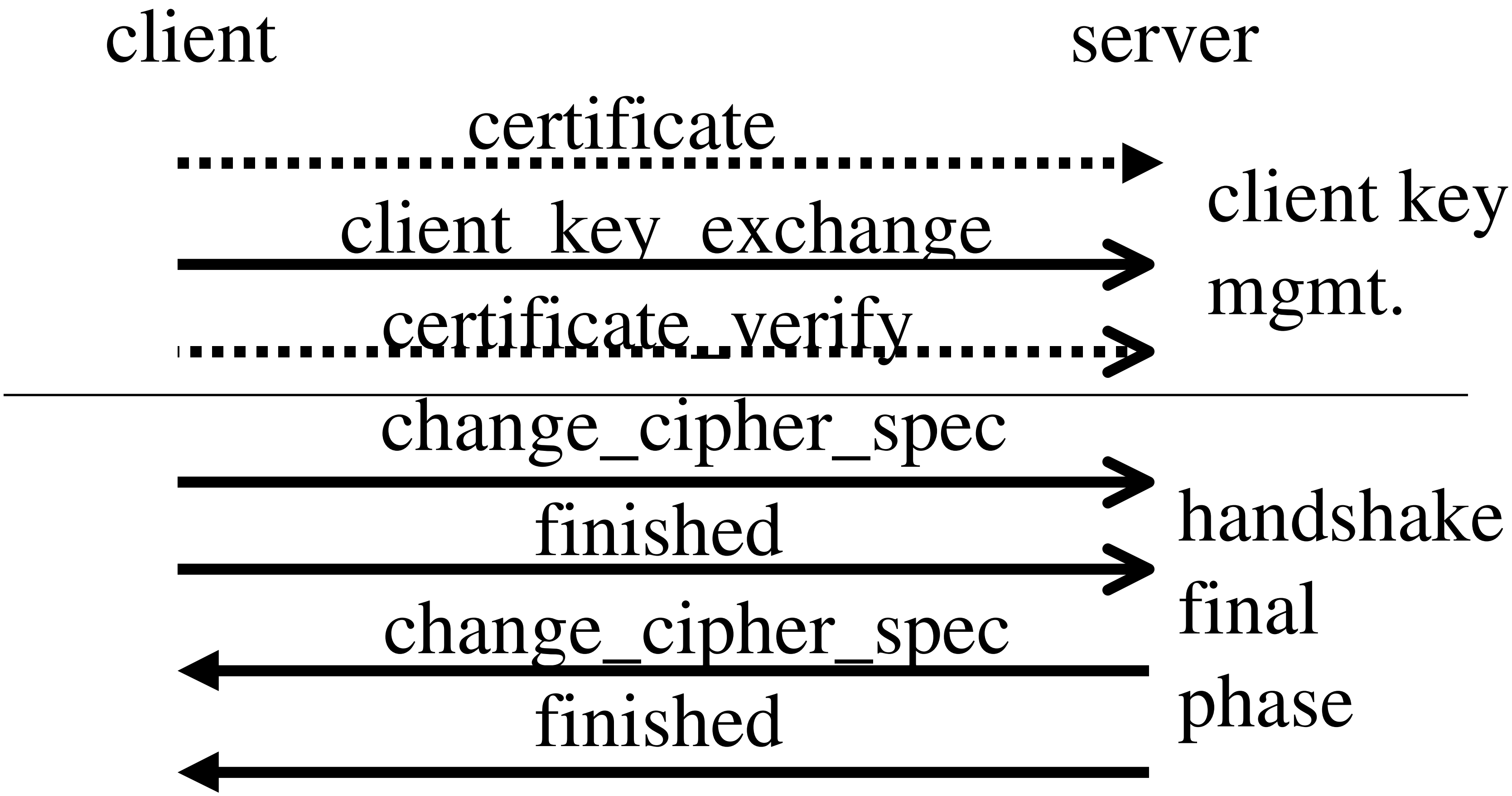
rough protocol exchange

- tcp socket client/server port 443 opened (https:www.jiminc.com)
- ssl handshake protocol occurs - establish session-key/cipher suite to be used
- ssl change_cipher_spec - finish initial session setup
- ssl record protocol used with http encapsulated inside it

handshake protocol



handshake protocol



hellos

- client_hello parameters:
  - version: ssl version
  - random: random values generated by client, timestamp + 28 bytes of randomness (nonce + anti-replay protection)
  - session ID: 0 means new session, else session update
  - cipher suite: list of possible sets of crypto transforms to be used, session key + crypto
  - compression method: list of compression methods client can do

server-side hello parameters

- same parameters however:
  - random field is server generated, not related to client random field
  - sessionID - if client field was zero, generated by server, else match client
  - cipherSuite - client proposes, server selects

cipherSuite parameters

- ciper suite contains a # of possibilities
- key exchange methods:
  - RSA used; i.e., public key encrypts session-key. certificate must be provided
  - Diffie-Hellman in several forms, fixed/ephemeral/anonymous (no authentication)
  - Fortezza (but not in IETF version)
- cipher spec specifies encryption/MAC
  - encryption: rc4, rc2, des, 3des, des40, idea
  - hash: md5/sha-1
  - other params, hash size (16/20), iv size, etc.

server-side authentication and key exchange

- server authentication phase - what happens depends on key exchange protocol
- if not anonymous DH, server sends 1 (or more) X.509 certificates in **certificate message**
  - signed by some CA, client assumed to have CA public key in CA certificate so it can verify certificate or may have server cert. already

server key exchange message

- sent if needed; .e.g, we need it for:
  - anonymous and ephemeral DH (latter includes signature)
- not needed for fixed DH or RSA key exchange to be used
- e.g., with ephemeral DH, we can now compute a one-time session key to be used with cipher spec algorithms

server certificate_request message

- server that is not using anonymous DH may now request certificate from client (user cert)
- includes two params: certificate type and certificate authorities e.g.,
  - RSA (sig only), DSS, and various forms for use with DH
- cert. authorities: list of DN acceptable to server

server_done message

- always sent - client may now take key materials and
- verify server key
  - else error
- initiate optional client authentication
  - if none, send no_certificate alert
- and key exchange phase

client authentication

- if server asks, client sends certificate message
- followed by client_key exchange message which is REQUIRED
  - if RSA, client generates pre-master secret and encrypts with server public key or temporary key from server_key_exchange
  - if DH ephemeral, send DH parameters

client authentication, cont.

- client may send **certificate_verify_message** as last part
- this is sent if client sent its own certificate which has signing capability
- client uses private key to sign parameters that server can use client's public key to verify
- thus server knows client has private/public key pair

final finish phase

- client sends change_cipher_spec message
  - means now using negotiated algorithms and keys
  - basically boolean
- client then sends finished msg using negotiated keys
- server does likewise, handshake now compete

some TLS differences

- fortezza dropped
- HMAC versions of md5/sha expected to be used

netscape 4.0 certificate download

- certificates may be imported into netscape
- X509 format including binary or text
  - DER encoded (binary)
  - PKCS #7 in several forms
  - text format if base64 encoded

    -------BEGIN CERTIFICATE----
    garp ...
    -------END CERTIFICATE----

MIME-based content types

- can be used to give netscape client certs via HTTP download
- **application/x-x509-user-cert** - it's a user certificate
  - private key must exist
- **application/x-x509-ca-cert** - CA certificate, may be cert chain of CAs
- **application/x-x509-email-cert** for S/MIME

netscape certificate types (client POV)

- CA certificate
  - self-signed by CA
  - if server cert shows up signed by CA, netscape verifies using CA public key and accepts
- site certificate (web server cert)
  - if not signed by CA, netscape asks user if he/she wants to trust it

netscape user cert process (e.g.)

- client connects to http://mysite/cert-request.html
- user prompted via form for relevant info (should be some process here for site admin to make sure user is user)
- html form contains <KEYGEN>
  - makes netscape create public/private keys
  - makes user create passphrase
  - server signed HTML public key returned to server

user cert. generation continued

- certificate issuer now has user public key, can begin certificate generation process
- creates locally signed (with local CA private key) user certificate
- sends email to user with URL for certificate
- user can use netscape now to download certificate

issues of trust

- server-side
  - public server server certificate needs to be signed by somebody else (so you can believe that somebody else is vouching for them ...)
  - self-signed certificate by Company X for Company X employees does not need this
- client-side
  - server-side encryption does not limit who uses the server ... therefore may want user authentication
  - if we have user-side certs, who should issue them?

user-side certs vs. passwords

- pro: certificate may go over network, ASCII password should not
- pro (and con ...): certificates are stored as files (don't have to remember)
  - but files can be lost
  - files are per computer
  - passphrase is important here and you have to remember it!
- pro: may be stronger form of authentication
  - this is veyr likely true

certificate extensions

- certificates can have X509 extensions and hence be customized for individual users or user groups
  - Clearance = Top Secret
  - might decide what part of web site particular user can see
  - strong argument for in-house certificate generation (if functionality needed)

import vs. export cert. services?

- external vendors for cert. services exist; e.g., VeriSign, BBN, etc.
- company might wish to do it in-house though, why?
  - tight control of management policies
  - use extensions
  - may or may not have better turn-around on needed services (delete this cert. fast ...)

what are trust models for?

- ipsec?
- ssh? - consider multi-user systems
- ssl?
  - consider e-commerce? (risk vs. trust)
- what kinds of interactions are possible?
- what kinds of interactions are NOT possible
- point: **you can distinguish nonsense vs. sense**