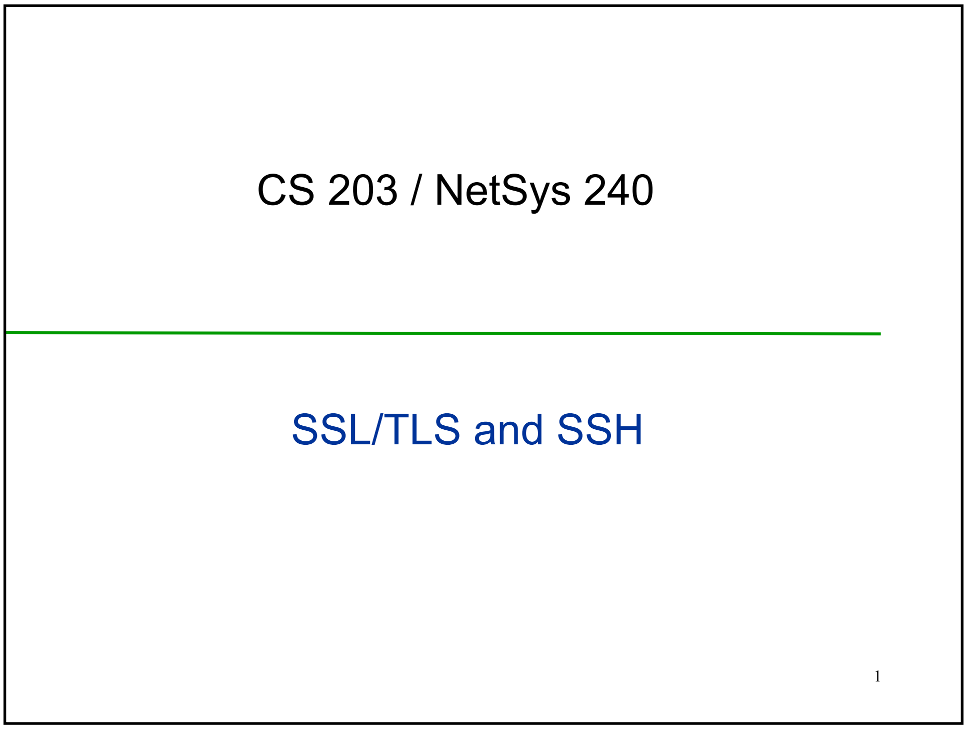
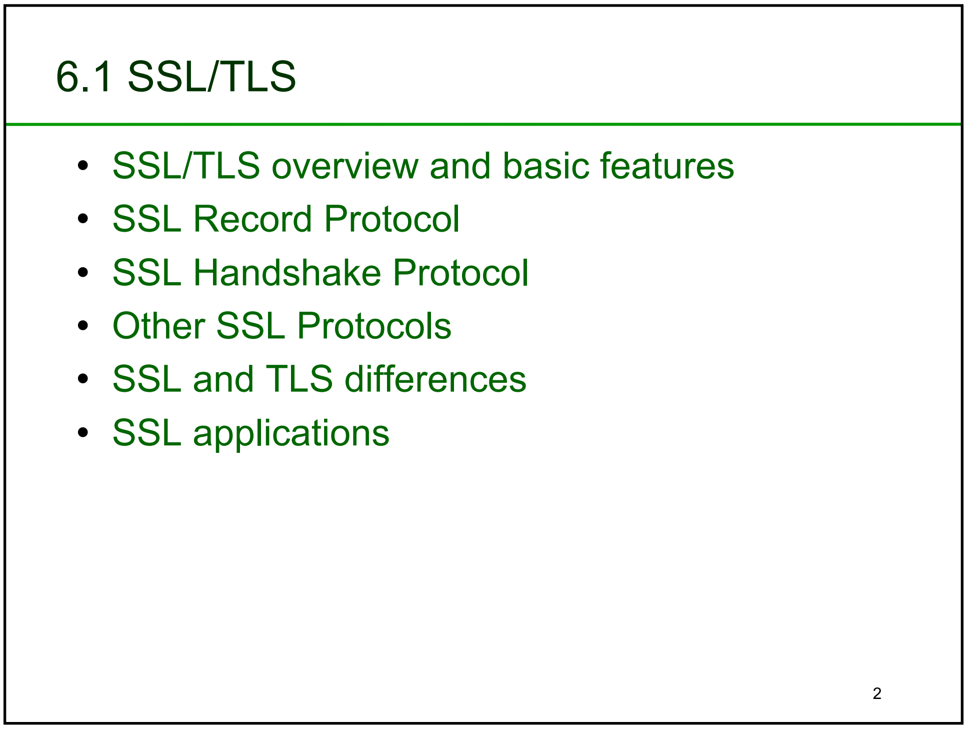


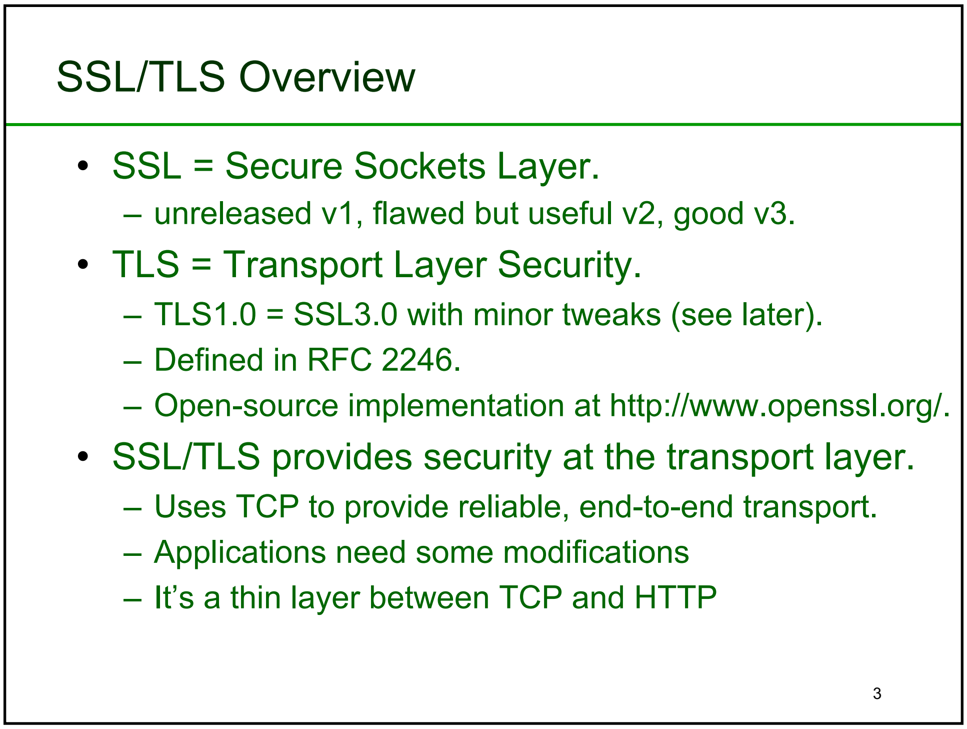
CS 203 / NetSys 240

SSL/TLS and SSH



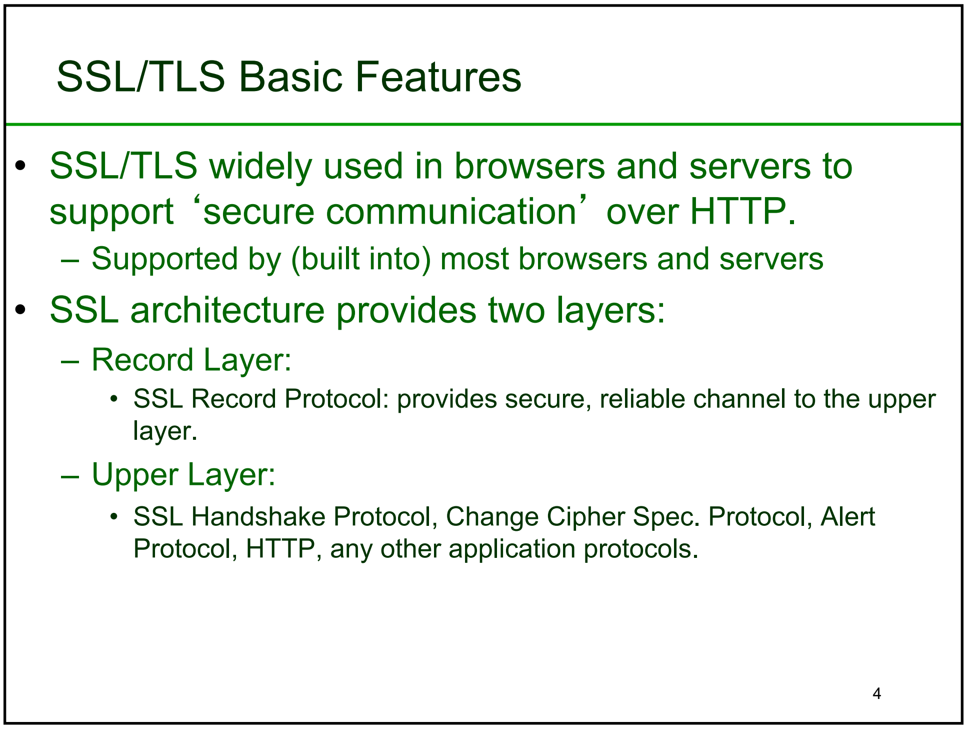
6.1 SSL/TLS

- SSL/TLS overview and basic features
- SSL Record Protocol
- SSL Handshake Protocol
- Other SSL Protocols
- SSL and TLS differences
- SSL applications



SSL/TLS Overview

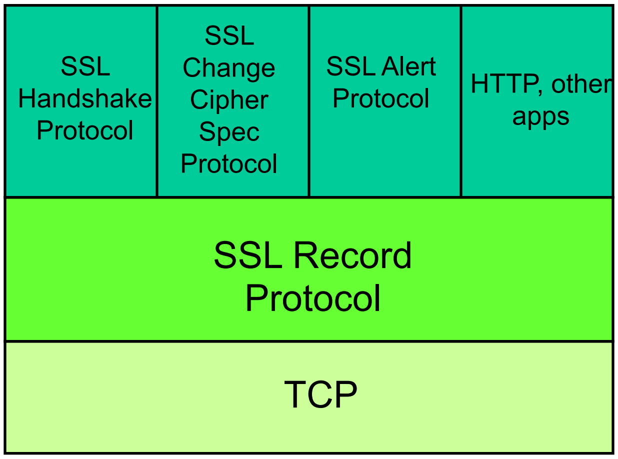
- SSL = Secure Sockets Layer.
  - unreleased v1, flawed but useful v2, good v3.
- TLS = Transport Layer Security.
  - TLS1.0 = SSL3.0 with minor tweaks (see later).
  - Defined in RFC 2246.
  - Open-source implementation at http://www.openssl.org/.
- SSL/TLS provides security at the transport layer.
  - Uses TCP to provide reliable, end-to-end transport.
  - Applications need some modifications
  - It's a thin layer between TCP and HTTP



SSL/TLS Basic Features

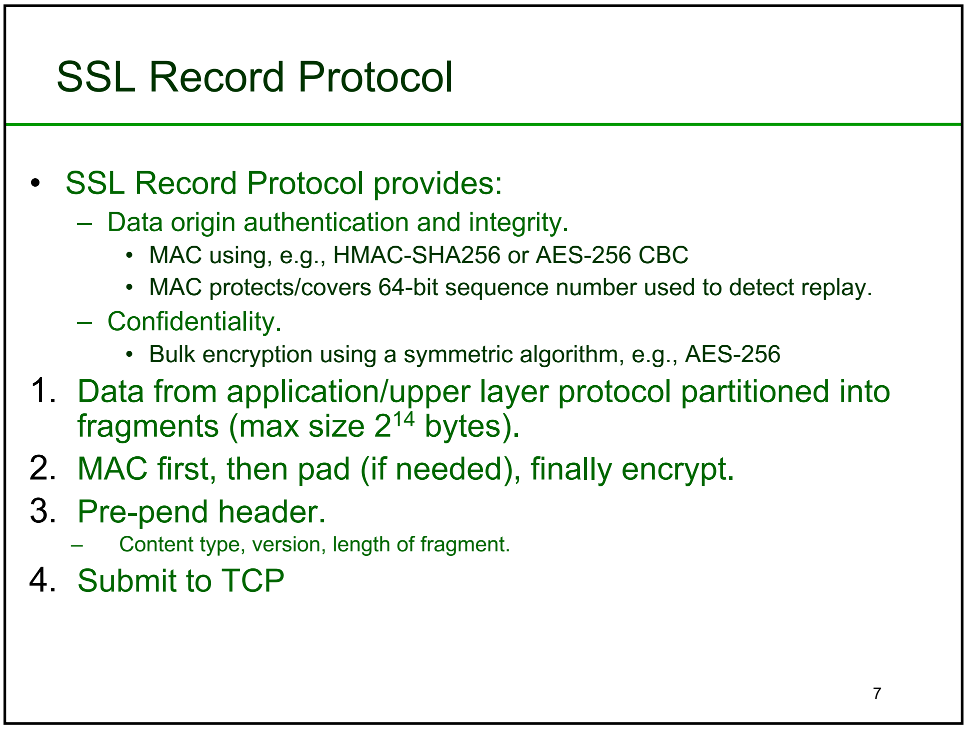
- SSL/TLS widely used in browsers and servers to support ‘secure communication’ over HTTP.
  - Supported by (built into) most browsers and servers
- SSL architecture provides two layers:
  - Record Layer:
    - SSL Record Protocol: provides secure, reliable channel to the upper layer.
  - Upper Layer:
    - SSL Handshake Protocol, Change Cipher Spec. Protocol, Alert Protocol, HTTP, any other application protocols.

SSL Protocol Architecture



SSL Record Protocol

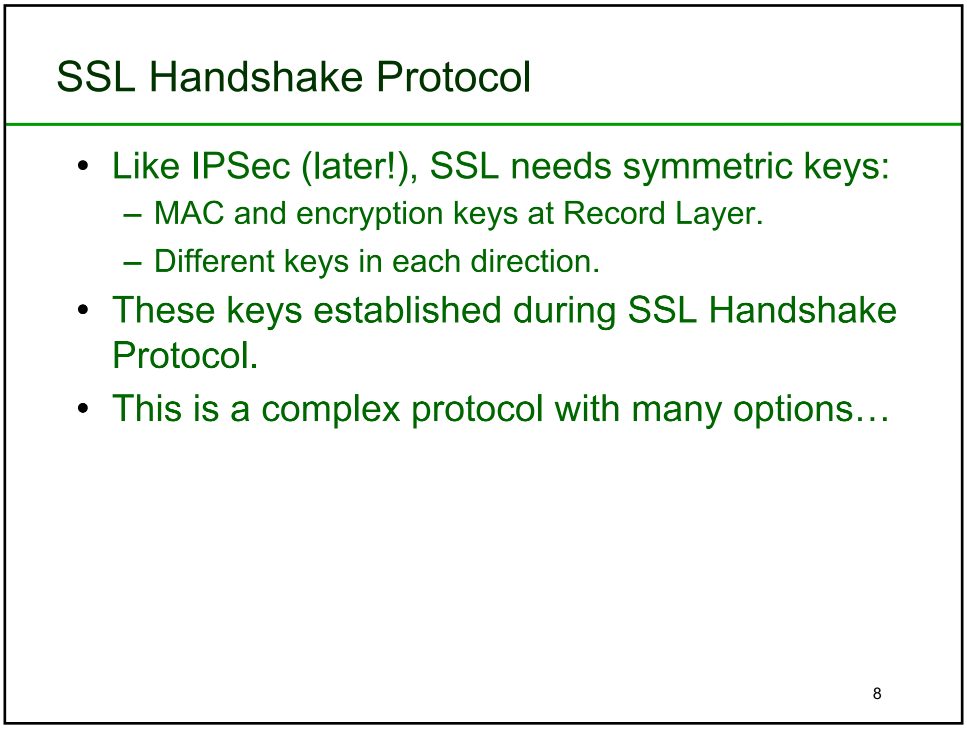
- Provides secure, reliable channel to the upper layer.
- Carries application data and SSL ‘management’ data.
- Session concept:
  - Sessions created by handshake protocol.
  - Defines set of cryptographic parameters (encryption and hash algorithms, master secret, certificates).
  - Carries multiple *connections* to avoid repeated use of expensive handshake protocol.
- Connection concept:
  - State defined by nonces, secret keys for MAC and encryption, IVs, sequence numbers.
  - Keys for many connections derived from single master secret created during handshake protocol.



SSL Record Protocol

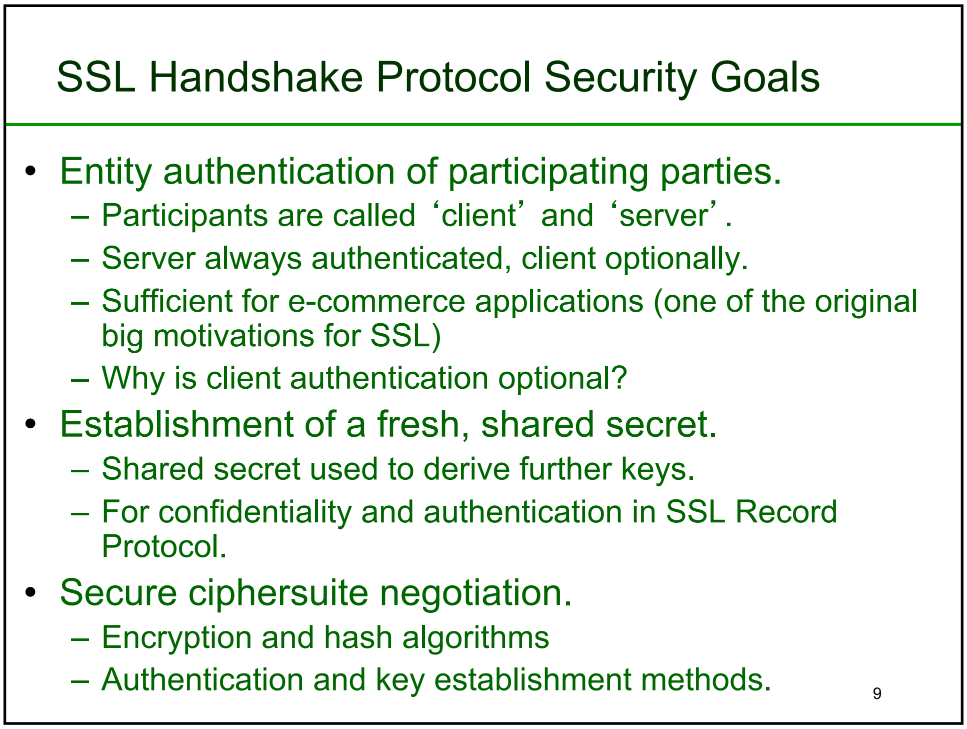
- SSL Record Protocol provides:
  - Data origin authentication and integrity.
    - MAC using, e.g., HMAC-SHA256 or AES-256 CBC
    - MAC protects/covers 64-bit sequence number used to detect replay.
  - Confidentiality.
    - Bulk encryption using a symmetric algorithm, e.g., AES-256

1. Data from application/upper layer protocol partitioned into fragments (max size 2^{14} bytes).
2. MAC first, then pad (if needed), finally encrypt.
3. Pre-pend header.
   - Content type, version, length of fragment.
4. Submit to TCP



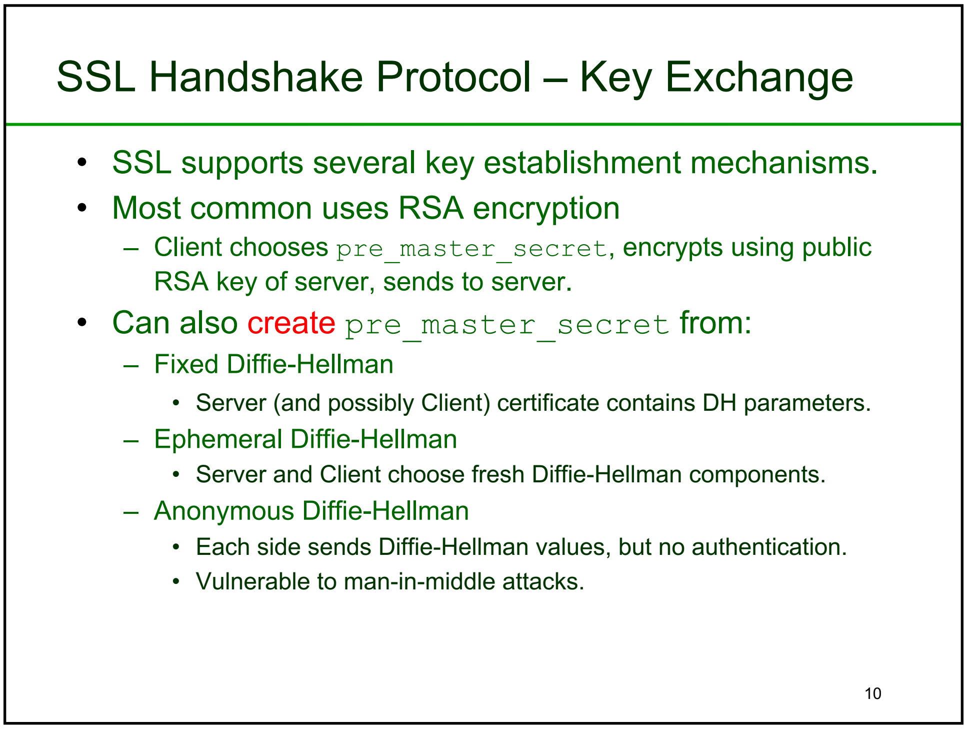
SSL Handshake Protocol

- Like IPSec (later!), SSL needs symmetric keys:
  - MAC and encryption keys at Record Layer.
  - Different keys in each direction.
- These keys established during SSL Handshake Protocol.
- This is a complex protocol with many options…



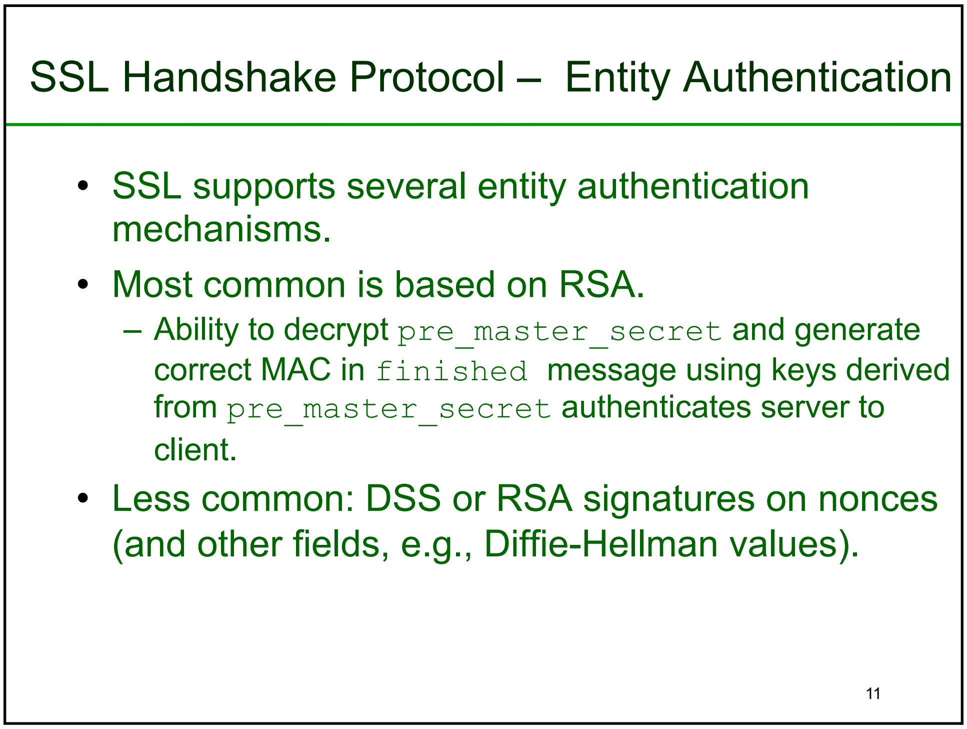
SSL Handshake Protocol Security Goals

- Entity authentication of participating parties.
  - Participants are called ‘client’ and ‘server’.
  - Server always authenticated, client optionally.
  - Sufficient for e-commerce applications (one of the original big motivations for SSL)
  - Why is client authentication optional?
- Establishment of a fresh, shared secret.
  - Shared secret used to derive further keys.
  - For confidentiality and authentication in SSL Record Protocol.
- Secure ciphersuite negotiation.
  - Encryption and hash algorithms
  - Authentication and key establishment methods.



SSL Handshake Protocol – Key Exchange

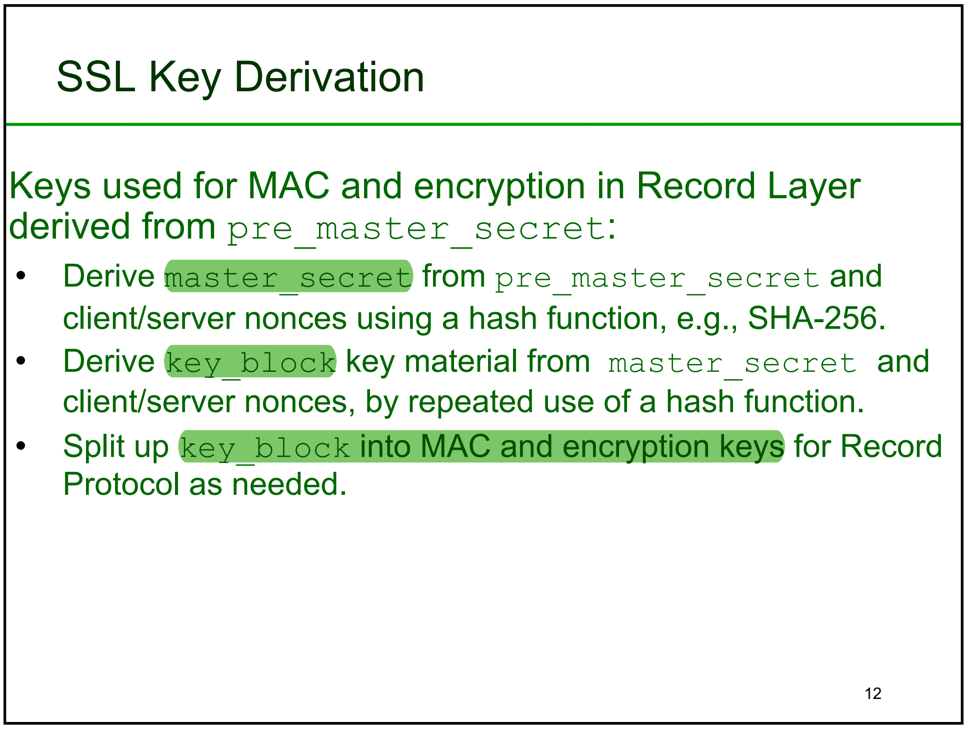
- SSL supports several key establishment mechanisms.
- Most common uses RSA encryption
  - Client chooses `pre_master_secret`, encrypts using public RSA key of server, sends to server.
- Can also create `pre_master_secret` from:
  - Fixed Diffie-Hellman
    - Server (and possibly Client) certificate contains DH parameters.
  - Ephemeral Diffie-Hellman
    - Server and Client choose fresh Diffie-Hellman components.
  - Anonymous Diffie-Hellman
    - Each side sends Diffie-Hellman values, but no authentication.
    - Vulnerable to man-in-middle attacks.



SSL Handshake Protocol – Entity Authentication

- SSL supports several entity authentication mechanisms.
- Most common is based on RSA.
  - Ability to decrypt `pre_master_secret` and generate correct MAC in `finished` message using keys derived from `pre_master_secret` authenticates server to client.
- Less common: DSS or RSA signatures on nonces (and other fields, e.g., Diffie-Hellman values).

11

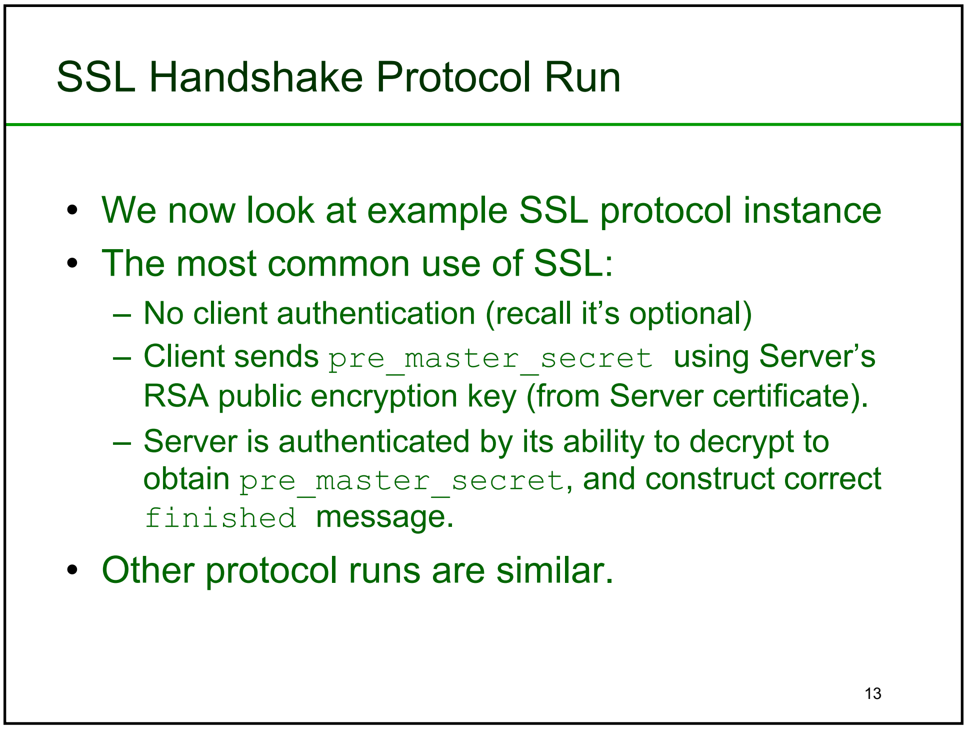


SSL Key Derivation

Keys used for MAC and encryption in Record Layer derived from `pre_master_secret`:

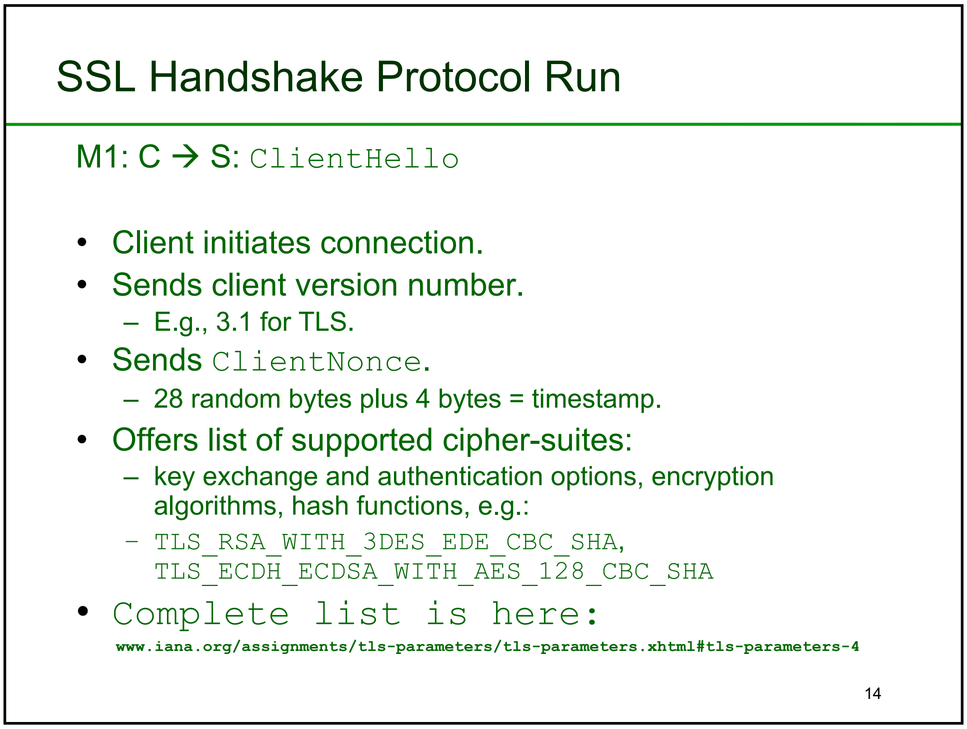
- Derive `master_secret` from `pre_master_secret` and client/server nonces using a hash function, e.g., SHA-256.
- Derive `key_block` key material from `master_secret` and client/server nonces, by repeated use of a hash function.
- Split up `key_block` into MAC and encryption keys for Record Protocol as needed.

12



SSL Handshake Protocol Run

- We now look at example SSL protocol instance
- The most common use of SSL:
  - No client authentication (recall it’s optional)
  - Client sends `pre_master_secret` using Server’s RSA public encryption key (from Server certificate).
  - Server is authenticated by its ability to decrypt to obtain `pre_master_secret`, and construct correct `finished` message.
- Other protocol runs are similar.

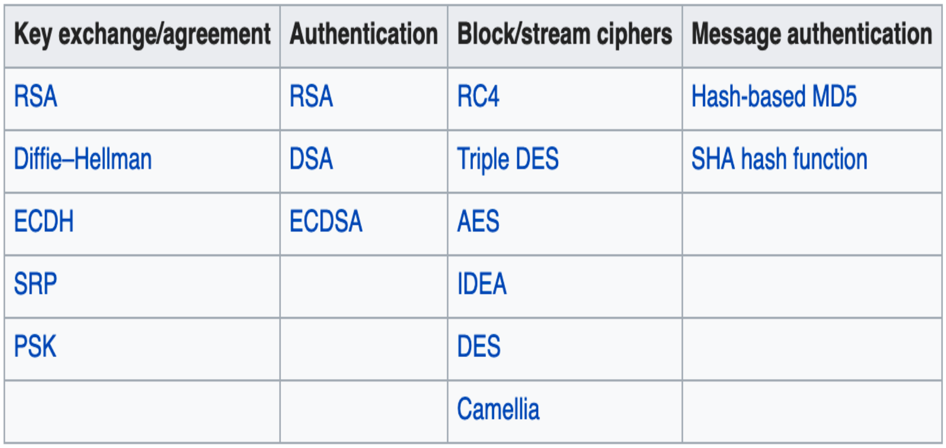


SSL Handshake Protocol Run

M1: C → S: `ClientHello`

- Client initiates connection.
- Sends client version number.
  - E.g., 3.1 for TLS.
- Sends `ClientNonce`.
  - 28 random bytes plus 4 bytes = timestamp.
- Offers list of supported cipher-suites:
  - key exchange and authentication options, encryption algorithms, hash functions, e.g.:
  - `TLS_RSA_WITH_3DES_EDE_CBC_SHA`, `TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA`
- `Complete list is here:`
  `www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4`

Supported Cipher Suites in TLS 1.0-1.2



Key exchange/agreement	Authentication	Block/stream ciphers	Message authentication
RSA	RSA	RC4	Hash-based MD5
Diffie–Hellman	DSA	Triple DES	SHA hash function
ECDH	ECDSA	AES	
SRP		IDEA	
PSK		DES	
		Camellia	

SSL Handshake Protocol Run

M2: S → C: `ServerHello, ServerCertChain, ServerHelloDone`

- `ServerHello message:`
  - Server version number.
  - `ServerNonce` and `SessionID`.
  - Selects single ciphersuite from list offered by client.
    - e.g., `TLS_RSA_WITH_3DES_EDE_CBC_SHA`.
- `ServerCertChain` message:
  - Allows client to validate server's public key back to acceptable root of trust.
- (optional) `CertRequest` message:
  - Omitted in this example – no client authentication.
- Finally, `ServerHelloDone`.

SSL Handshake Protocol Run

M3: C → S: `ClientKeyExchange`, `ChangeCipherSpec`, `ClientFinished`

- `ClientKeyExchange` contains encryption of `pre_master_secret` under server's public key.
- `ChangeCipherSpec` indicates that client is updating cipher suite to be used on this session.
  - Sent using SSL Change Cipher Spec Protocol.
- (optional) `ClientCertificate`, `ClientCertificateVerify` messages.
  - Only when client is authenticated.
- Finally, `ClientFinished` message.
  - MAC on all messages sent so far (both sides).
  - MAC computed using `master_secret`.

SSL Handshake Protocol Run

M4: S → C: `ChangeCipherSpec`, `ServerFinished`

- `ChangeCipherSpec` indicates that server is updating cipher suite to be used for this session.
  - Sent using SSL Change Cipher Spec. Protocol.
- Finally, `ServerFinished` message.
  - MAC on all messages sent so far (both sides).
  - MAC computed using `master_secret`.
  - Server can only compute MAC if it can decrypt `pre_master_secret` in M3.

SSL Handshake Protocol Run

Summary:

M1: C → S: `ClientHello`

M2: S → C: `ServerHello, ServerCertChain,ServerHelloDone`

M3: C → S: `ClientKeyExchange, ChangeCipherSpec, ClientFinished`

M4: S → C: `ChangeCipherSpec, ServerFinished`

SSL Handshake Protocol Run

1. Is the client authenticated to the server in this protocol run?
2. Can an adversary learn the value of `pre_master_secret`?
3. Is the server authenticated to the client?

1. No!
2. No! Client has validated server's public key; Only holder of private key can decrypt `ClientKeyExchange` to learn `pre_master_secret`.
3. Yes! `ServerFinished` includes MAC on nonces computed using key derived from `pre_master_secret`.

Other SSL Handshake Protocol Runs

- Many optional/situation-dependent protocol messages:
  - M2 (S→C) can include:
    - `ServerKeyExchange` (e.g. for DH key exchange).
    - `CertRequest` (for client authentication).
  - M3 (C→S) can include:
    - `ClientCert` (for client authentication),
    - `ClientCertVerify` (for client authentication).
- For details, see Stallings Figure 7.6 and pp. 212-219 (SSL) or RFC 2246 (TLS).

SSL Handshake Protocol – Additional Features

- SSL Handshake Protocol supports *session resumption* and *ciphersuite re-negotiation*.
  - Allows authentication and shared secrets to be reused across multiple connections.
    - e.g., to fetch another webpage from same website.
  - Allows re-keying of current connection using fresh nonces.
  - Allows change of ciphersuite during session.
  - `ClientHello` quotes old `SessionID`.
  - Both sides contribute new nonces, update `master_secret` and `key_block`.
  - All protected by existing Record Protocol.

Other SSL Protocols

- Alert protocol.
  - Management of SSL session, error messages.
  - Fatal errors and warnings.
- Change cipher spec protocol.
  - Not part of SSL Handshake Protocol.
  - Used to indicate that entity is changing to recently agreed ciphersuite.
- Both protocols run over Record Protocol (so peers of Handshake Protocol).

SSL and TLS

TLS1.0 = SSL3.0 with minor differences.

- TLS signalled by version number 3.1.
- Use of HMAC for MAC algorithm.
- Different method for deriving keying material (`master-secret` and `key-block`).
  - Pseudo-random function based on HMAC with MD5 and SHA-1.
- Additional alert codes.
- More client certificate types.
- Variable length padding.
  - Can be used to hide lengths of short messages and so frustrate traffic analysis.
- And more ….

SSL/TLS Applications, e.g., e-commerce

- Secure e-commerce using SSL/TLS.
  - Client authentication not needed until client decides to buy something.
  - SSL provides secure channel for sending credit card information, personal details, etc. (TO THE RIGHT SERVER)
  - Client authenticated using credit card information, merchant bears (most of) risk.
  - Very widespread since late 1990-s

SSL/TLS Applications

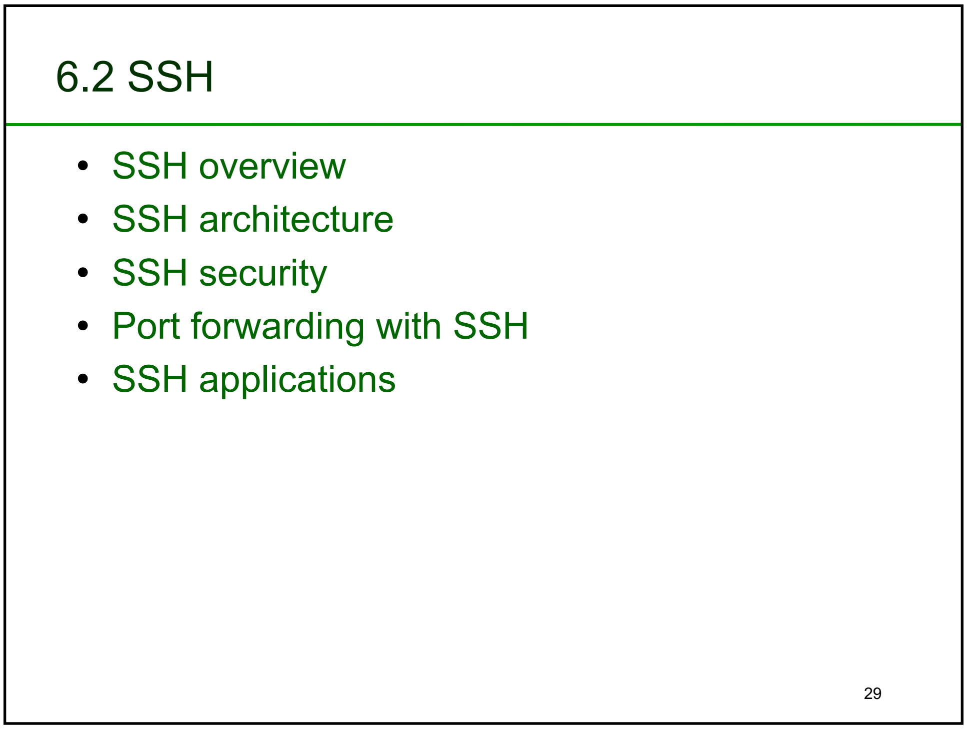
- Secure e-commerce: some issues.
  - No guarantees about what happens to client data (including credit card details) after session: may be stored on insecure server.
  - Does client understand meaning of certificate expiry and other security warnings?
  - Does client software *actually* check complete certificate chain?
  - Does the name in certificate match the URL of e-commerce site? Does the user check this?
  - Is the site the one the client thinks it is?
  - Is the client software proposing appropriate ciphersuites?

SSL/TLS Applications

- Secure electronic banking.
  - Client authentication may be enabled using client certificates.
    - Issues of registration, secure storage of private keys, revocation and re-issue.
  - Otherwise, SSL provides secure channel for sending username, password, mother's maiden name,…
    - What else does client use same password for?
  - Does client understand meaning of certificate expiry and other security warnings?
  - Is client software proposing appropriate ciphersuites?
    - Enforce from server.

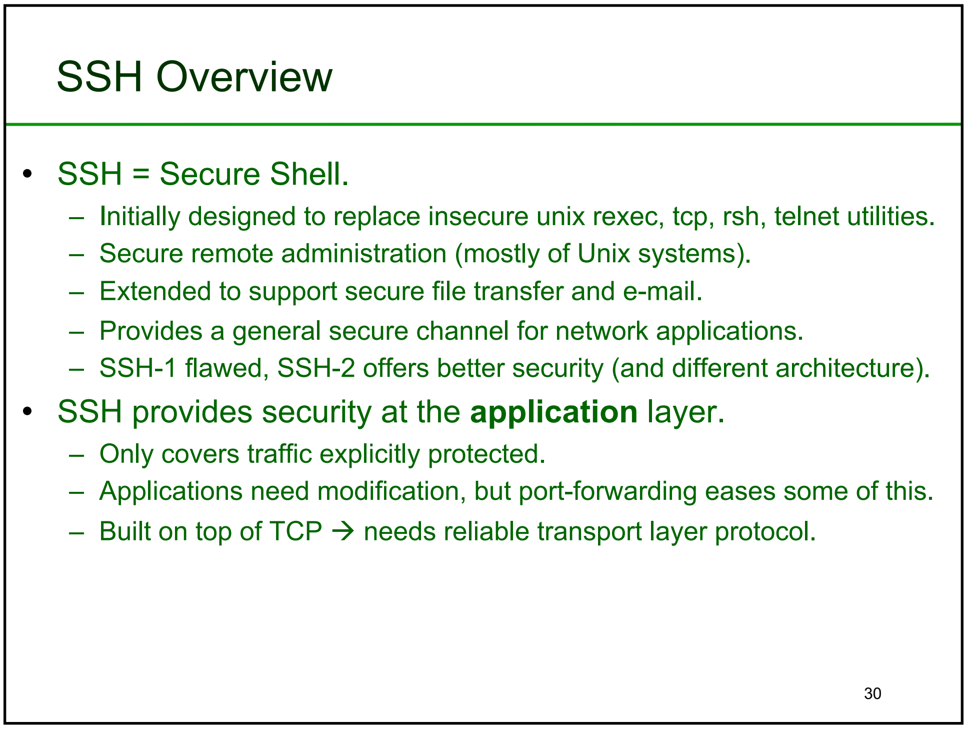
Some SSL/TLS Security Flaws

- (Historical) flaws in random number generation for SSL.
  - Low quality RNG leads to predictable session keys.
  - Goldberg and Wagner, Dr. Dobb's Journal, Jan. 1996.
  - http://www.ddj.com/documents/s=965/ddj9601h/
- Flaws in error reporting.
  - (differing response times by server in event of padding failure and MAC failure) + (analysis of padding method for CBC-mode) = recovery of SSL plaintext.
  - Canvel, Hiltgen, Vaudenay and Vuagnoux, Crypto 2003.
  - http://lasecwww.epfl.ch/php_code/publications/search.php?ref=CHVV03
- Timing attacks.
  - analysis of OpenSSL server response times allows attacker in same LAN segment to derive server's private key!
  - Boneh and Brumley, 12th Usenix Security Symposium, 2003.
  - http://crypto.stanford.edu/~dabo/abstracts/ssl-timing.html



6.2 SSH

- SSH overview
- SSH architecture
- SSH security
- Port forwarding with SSH
- SSH applications



SSH Overview

- SSH = Secure Shell.
  - Initially designed to replace insecure unix rexec, tcp, rsh, telnet utilities.
  - Secure remote administration (mostly of Unix systems).
  - Extended to support secure file transfer and e-mail.
  - Provides a general secure channel for network applications.
  - SSH-1 flawed, SSH-2 offers better security (and different architecture).
- SSH provides security at the **application** layer.
  - Only covers traffic explicitly protected.
  - Applications need modification, but port-forwarding eases some of this.
  - Built on top of TCP → needs reliable transport layer protocol.

SSH Overview

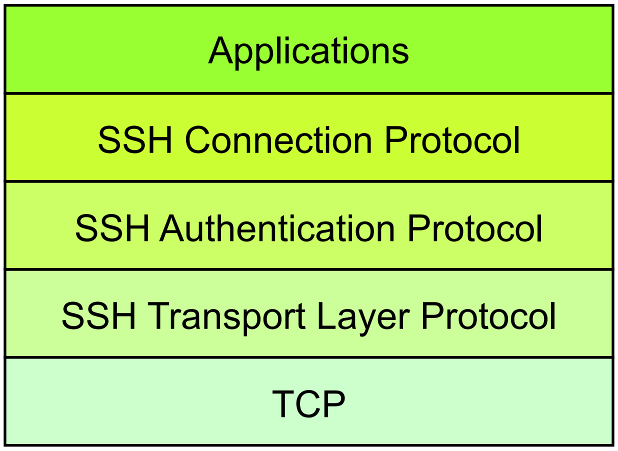
- SSH Communications Security (SCS).
  - www.ssh.com.
  - Founded by Tatu Ylonen, author of SSH-1.
  - SSH is a trademark of SCS.
- Open source version from OpenSSH.
- IETF Secure Shell (SECSH) working group.
  - Standards body for SSH:
    www.ietf.org/html.charters/secsh-charter.html
- Long-running confusion and dispute over naming.

SSH-2 Architecture

SSH-2 adopts a three-layer architecture:

- SSH Transport Layer Protocol.
  - Initial connection.
  - Server authentication (almost always).
  - Sets up secure channel between client and server.
- SSH Authentication Protocol
  - Client authentication over secure transport layer channel.
- SSH Connection Protocol
  - Supports multiple connections over a single transport layer protocol secure channel.
  - Efficiency (session re-use).

SSH-2 Architecture


Applications

SSH Connection Protocol

SSH Authentication Protocol

SSH Transport Layer Protocol

TCP

SSH-2 Security Goals

- Server (nearly) always authenticated in transport layer protocol.
- Client (nearly) always authenticated in authentication protocol.
  - By public key (DSS, RSA, SPKI, OpenPGP).
  - Or simple password for particular application over secure channel.
- Establishment of a fresh, shared secret.
  - Shared secret used to derive further keys, similar to SSL/IPSec.
  - For confidentiality and authentication in SSH transport layer protocol.
- Secure ciphersuite negotiation.
  - Encryption, MAC, and compression algorithms.
  - Server authentication and key exchange methods.

SSH-2 Algorithms

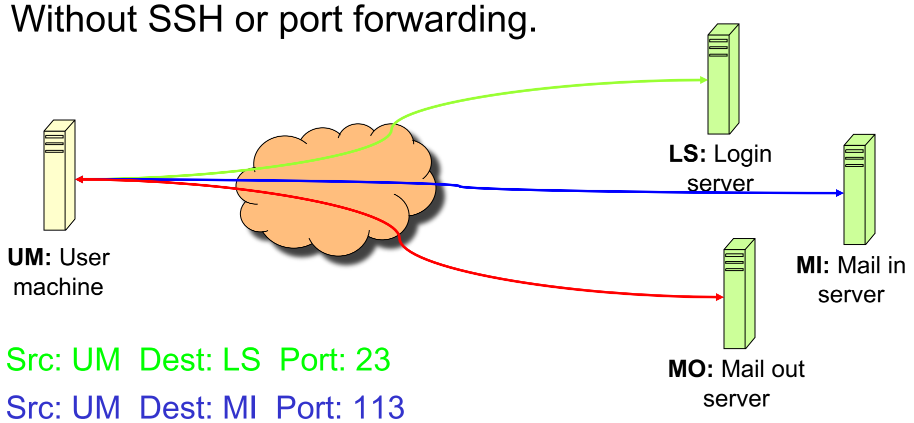
- Key establishment through Diffie-Hellman key exchange.
  - Variety of groups supported.
- Server authentication via RSA or DSS signatures on nonces (and other fields).
- HMAC-SHA1 or HMAC-MD5 for MAC algorithm.
- 3DES, RC4, or AES
- Pseudo-random function for key derivation.
- Small number of ‘official’ algorithms with simple DNS-based naming of ‘private’ methods.

SSH-1 versus SSH-2

- Many vulnerabilities have been found in SSH-1 .
  - SSH-1 Insertion attack exploiting weak integrity mechanism (CRC-32) and unprotected packet length field.
  - SSHv1.5 session key retrieval attack (theoretical).
  - Man-in-the-middle attacks (using e.g. dsniff).
  - DoS attacks.
    - Overload server with connection requests.
    - Buffer overflows.
- But SSH-1 widely deployed.
- And SSH-1 supports:
  - Wider range of client authentication methods (.rhosts and Kerberos).
  - Wider range of platforms.

SSH Port Forwarding

Without SSH or port forwarding.



Src: UM Dest: LS Port: 23

Src: UM Dest: MI Port: 113

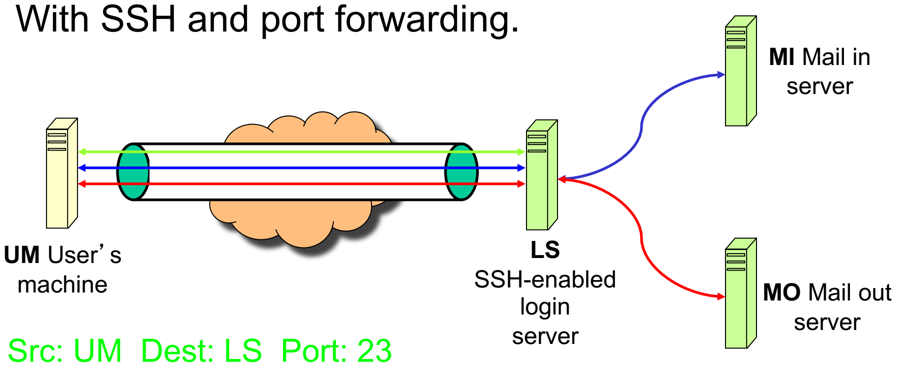
Src: UM Dest: MO Port: 25

SSH Port Forwarding

- Recall: TCP port number ‘identifies’ application.
- SSH on local machine:
  - Intercepts traffic bound for server.
  - Translates standard TCP port numbers.
    - E.g. port 113 → port 5113.
  - Sends packets to SSH-enabled server through SSH secure channel.
- SSH-enabled server:
  - Receives traffic.
  - Re-translates port numbers.
    - E.g. port 5113 → port 113.
  - Forwards traffic to appropriate server using internal network.

SSH Port Forwarding

With SSH and port forwarding.



Src: UM Dest: LS Port: 23

Src: UM Dest: MI Port: 113
Src: UM Dest: LS Port: 5113
Src: LS Dest: MI Port: 113

Src: UM Dest: MO Port: 25
Src: UM Dest: LS Port: 5025
Src: LS Dest: MO Port: 25

SSH Applications

- Anonymous ftp for software updates, patches...
  - No client authentication needed, but clients want to be sure of origin and integrity of software.
- Secure ftp.
  - E.g.upload of webpages to webserver using sftp.
  - Server now needs to authenticate clients.
  - Username and password may be sufficient, transmitted over secure SSH transport layer protocol.
- Secure remote administration.
  - SysAdmin (client) sets up terminal on remote machine.
  - SysAdmin password protected by SSH transport layer protocol.
  - SysAdmin commands protected by SSH connection protocol.
- Guerilla Virtual Private Network.
  - E.g. use SSH + port forwarding to secure e-mail communications.

6.3 Comparing IPSec, SSL/TLS, SSH

- All three have initial (authenticated) key establishment then key derivation.
  - IKE in IPSec
  - Handshake Protocol in SSL/TLS (can be unauthenticated!)
  - Authentication Protocol in SSH
- All protect ciphersuite negotiation.
- All three use keys established to build a ‘secure channel’.

Comparing IPSec, SSL/TLS, SSH

- Operate at different network layers.
  - This brings pros and cons for each protocol suite.
  - Recall `Where shall we put security?’ discussion.
  - Naturally support different application types, can all be used to build VPNs.
- All practical, but not simple.
  - Complexity leads to vulnerabilities.
  - Complexity makes configuration and management harder.
  - Complexity can create computational bottlenecks.
  - Complexity necessary to give both flexibility and security.

Comparing IPSec, SSL/TLS, SSH

Security of all three undermined by:

- Implementation weaknesses.
- Weak server platform security.
  - Worms, malicious code, rootkits,…
- Weak user platform security.
  - Keystroke loggers, malware,…
- Limited deployment of certificates and infrastructure to support them.
  - Especially client certificates.
- Lack of user awareness and education.
  - Users click-through on certificate warnings.
  - Users fail to check URLs.
  - Users send sensitive account details to bogus websites (“phishing”) in response to official-looking e-mail.

Secure Protocols – Last Words

A (mis)quote from Gene Spafford:

> “*Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit-card information from someone living in a cardboard box to someone living on a park bench.*”