Transport Level Security

CSS322: Security and Cryptography

Sirindhorn International Institute of Technology
Thammasat University

Prepared by Steven Gordon on 28 October 2013
css322y13s2l12, Steve/Courses/2013/s2/css322/lectures/transport.tex, r2965

Contents

Web Security Issues

TLS/SSL

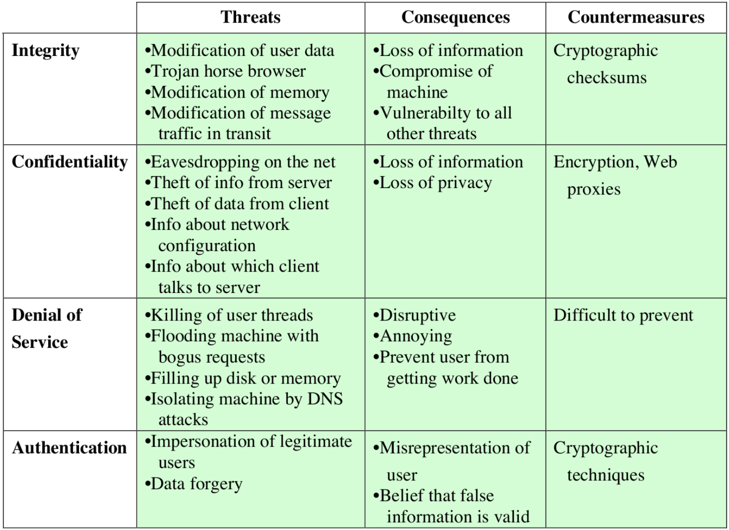
HTTPS

Secure Shell

Web Security Issues

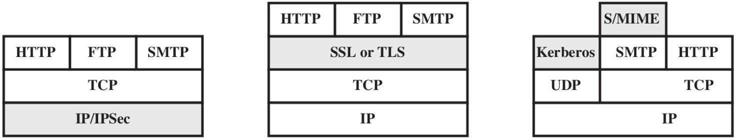
- Original Internet protocols do not have built-in security (IP, TCP, HTTP, . . .)
- Many threats arise for web and other Internet applications
- Issues at: client, server and traffic between client and server
- Cover: SSL/TLS, SSH, IPsec

Comparison of Threats on the Web



	Threats	Consequences	Countermeasures
Integrity	•Modification of user data •Trojan horse browser •Modification of memory •Modification of message traffic in transit	•Loss of information •Compromise of machine •Vulnerabilty to all other threats	Cryptographic checksums
Confidentiality	•Eavesdropping on the net •Theft of info from server •Theft of data from client •Info about network configuration •Info about which client talks to server	•Loss of information •Loss of privacy	Encryption, Web proxies
Denial of Service	•Killing of user threads •Flooding machine with bogus requests •Filling up disk or memory •Isolating machine by DNS attacks	•Disruptive •Annoying •Prevent user from getting work done	Difficult to prevent
Authentication	•Impersonation of legitimate users •Data forgery	•Misrepresentation of user •Belief that false information is valid	Cryptographic techniques

Security Options in TCP/IP



HTTP | FTP | SMTP
TCP
IP/IPSec

(a) Network Level

HTTP | FTP | SMTP
SSL or TLS
TCP
IP

(b) Transport Level

S/MIME
Kerberos | SMTP | HTTP
UDP | TCP
IP

(c) Application Level

- IPsec: Security for IP datagrams; general solution for all Internet traffic; implemented in OS
- SSL/TLS: Security for TCP segments; general solution for all TCP-based applications; implemented in libraries/applications (e.g. OpenSSL)
- Application-specific: Security for application messages; specific to each applications; implemented in single application

Contents

Web Security Issues

TLS/SSL

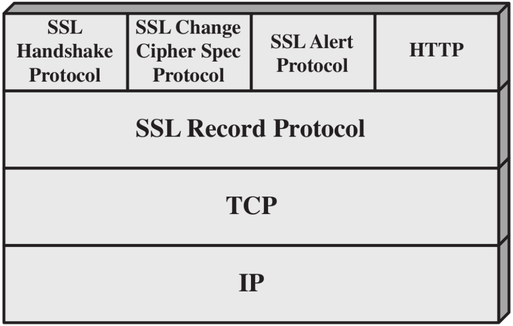
HTTPS

Secure Shell

SSL and TLS

- Secure Sockets Layer (SSL) originated in Netscape web browser
- Transport Layer Security (TLS) standardised by IETF
- SSLv3 and TLS are almost the same
- SSL provides security services to application layer protocols using TCP
- SSL architecture consists of multiple protocols

SSL Architecture



Record: provides confidentiality and message integrity

Handshake: authenticate entities, negotiate parameter values

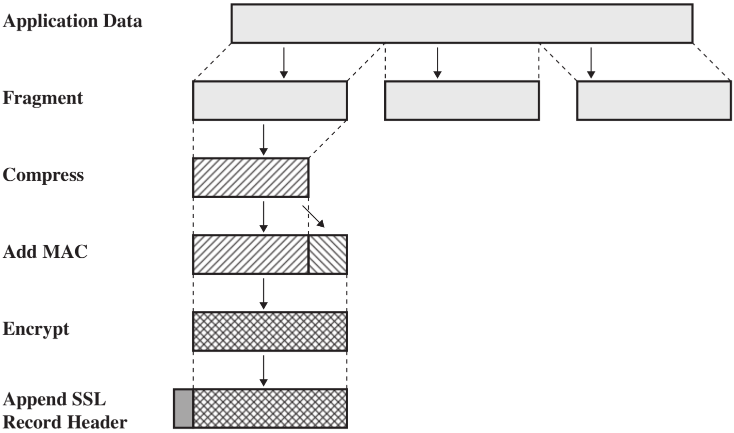
Change Cipher: change cipher for use in connection

Alert: alert peer entity of status/warning/error

Connections and Sessions

- SSL connection corresponds with TCP connection
  - Client and server may have multiple connections
- SSL session is association between client and server
  - Session created with Handshake protocol
  - Multiple connections can be associated with one session
  - Security parameters for session can be shared for connections
- State information is stored after Handshake protocol
  - Session: ID, certificate, compression, cipher spec, master secret, . . .
  - Connection: random values, encrypt keys, MAC secrets, IV, sequence numbers, . . .

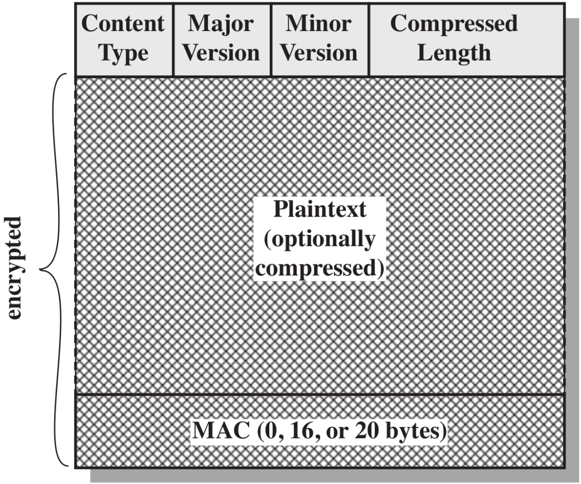
SSL Record Protocol Operation



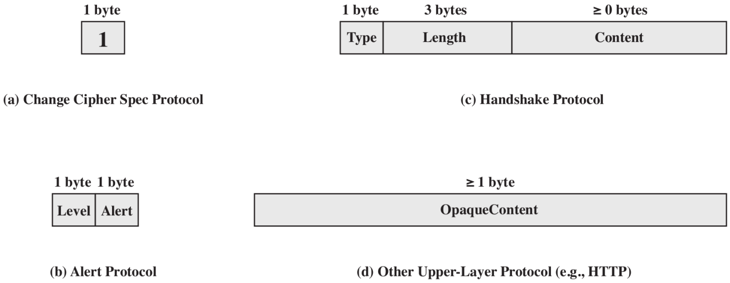
SSL Record Protocol

- Fragmentation: maximum fragment size is 16384 Bytes
- Compression: lossless; algorithm chosen in Handshake
- MAC: HMAC applied on compressed data; MAC secret key for connection used; MAC appended to compressed fragment
- Encrypt: applied to compressed fragment and MAC; algorithm chosen in Handshake
- SSL record header:
    - Content type: higher layer protocol (change cipher spec, alert, handshake, application)
    - Version
    - Compressed length in bytes

SSL Record Format



SSL Record Protocol Payload



(a) Change Cipher Spec Protocol

(c) Handshake Protocol

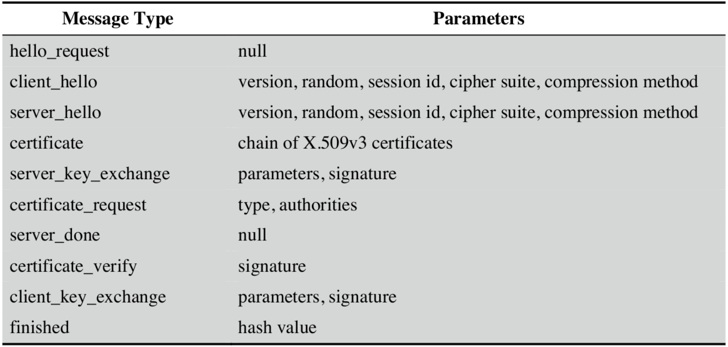
(b) Alert Protocol

(d) Other Upper-Layer Protocol (e.g., HTTP)

SSL Handshake Protocol

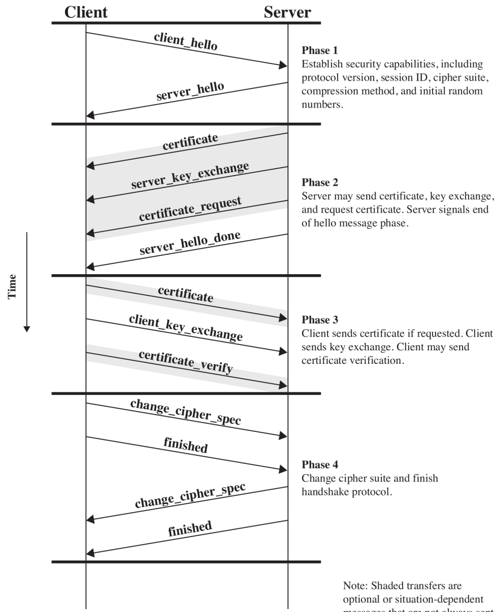
- Allow client and server to authenticate each other
- Negotiate encryption and MAC algorithms, exchange keys
  - Key Exchange: RSA, Diffie-Hellman
  - MAC: HMAC using SHA or MD5
  - Encryption: RC4, RC2, DES, 3DES, IDEA, AES
- Multiple phases:
  1. Establish security capabilities: client proposes algorithms, server selects one
  2. Server authentication and key exchange
  3. Client authentication and key exchange
  4. Finish setting up connection

SSL Handshake Protocol Messages



Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

SSL Handshake Protocol Operation



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Contents

Web Security Issues

TLS/SSL

HTTPS

Secure Shell

HTTPS

- HTTPS: HTTP over SSL (or TLS)
- URL uses https://
- Web server listens on port 443
- Encrypt: URL of requested document, contents of document, contents of browser forms, cookies, contents of HTTP header
- Server is authenticated using certificate (using SSL)
- Client is authenticated using password (using HTTP)

Contents

Web Security Issues

TLS/SSL

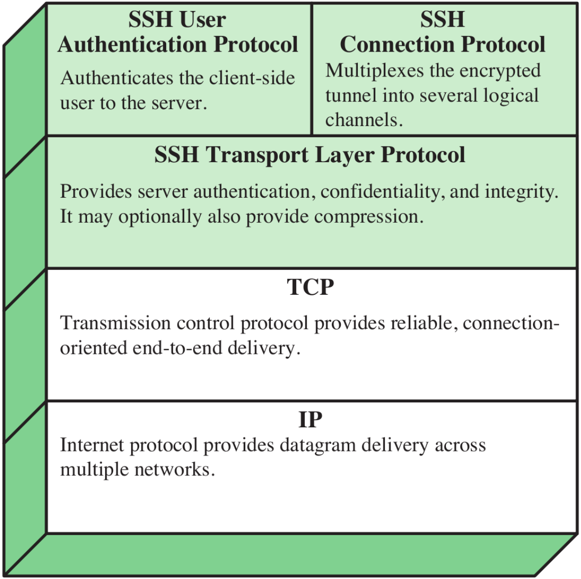
HTTPS

Secure Shell

Secure Shell

- TELNET provides a remote login facility; insecure
- Secure Shell (SSH) designed for secure remote login
- SSH also supports secure file transfer and tunnelling
- SSHv2 developed by IETF
- SSH architecture consists of 3 protocols

SSH Protocol Stack



SSH User Authentication Protocol

Authenticates the client-side user to the server.

SSH Connection Protocol

Multiplexes the encrypted tunnel into several logical channels.

SSH Transport Layer Protocol

Provides server authentication, confidentiality, and integrity. It may optionally also provide compression.

TCP

Transmission control protocol provides reliable, connection-oriented end-to-end delivery.

IP

Internet protocol provides datagram delivery across multiple networks.

SSH Authentication

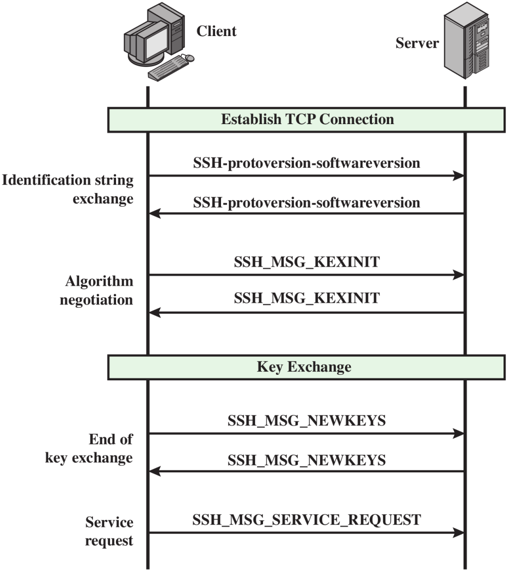
Server Authentication

- Server has public/private key pair
- Assume client knows server's public key
- During key exchange, server signs message with public key

Client Authentication

- Key-based: client has public/private key pair; server knows client public key
- Password-based: client sends password (encrypted); server knows password

SSH Transport Layer Packet Exchange



SSH Transport Layer Protocol

- Identification string exchange: each entity identifies protocol and software version
- Algorithm negotiation: client and server send list of supported algorithms, in order of preference; first common algorithm chosen
- Key exchange: Diffie-Hellman
- End of key exchange: new keys generated from shared secret, e.g.

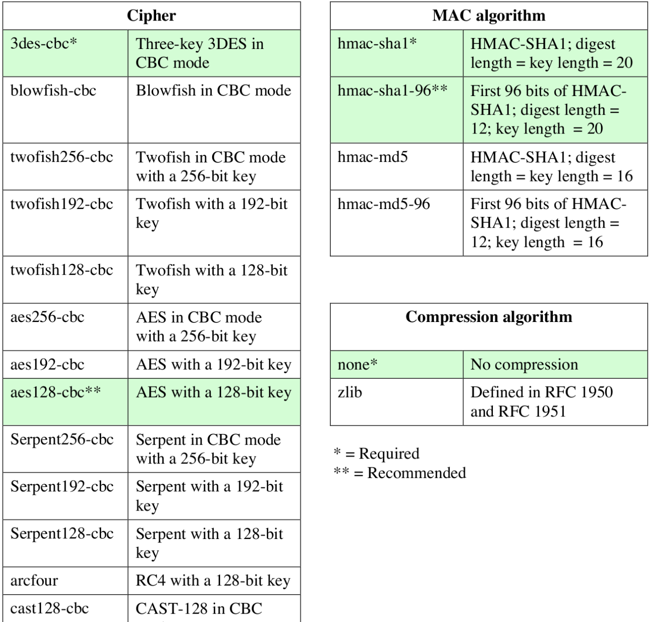
$$K_{c2s} = Hash(K||H||'C'||session\_id)$$

  where

$$H = Hash(ID_C||ID_C||M_C||M_S||PU_S||Y_A||Y_B||K)$$

- Service request for User Authentication or Connection Protocol

SSH Algorithms



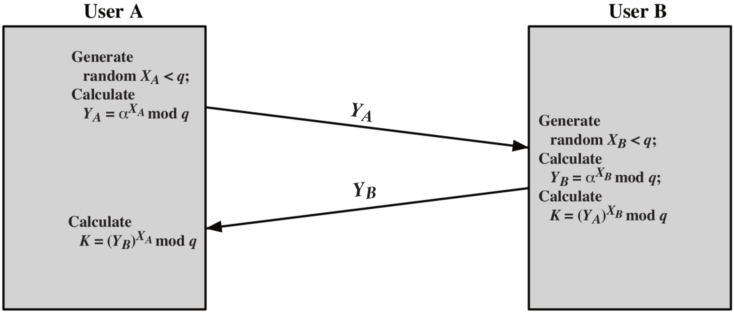
Cipher	
3des-cbc*	Three-key 3DES in CBC mode
blowfish-cbc	Blowfish in CBC mode
twofish256-cbc	Twofish in CBC mode with a 256-bit key
twofish192-cbc	Twofish with a 192-bit key
twofish128-cbc	Twofish with a 128-bit key
aes256-cbc	AES in CBC mode with a 256-bit key
aes192-cbc	AES with a 192-bit key
aes128-cbc**	AES with a 128-bit key
Serpent256-cbc	Serpent in CBC mode with a 256-bit key
Serpent192-cbc	Serpent with a 192-bit key
Serpent128-cbc	Serpent with a 128-bit key
arcfour	RC4 with a 128-bit key
cast128-cbc	CAST-128 in CBC

MAC algorithm	
hmac-sha1*	HMAC-SHA1; digest length = key length = 20
hmac-sha1-96**	First 96 bits of HMAC-SHA1; digest length = 12; key length = 20
hmac-md5	HMAC-SHA1; digest length = key length = 16
hmac-md5-96	First 96 bits of HMAC-SHA1; digest length = 12; key length = 16

Compression algorithm	
none*	No compression
zlib	Defined in RFC 1950 and RFC 1951

* = Required
** = Recommended

Key Exchange with Diffie-Hellman


User A
Generate random $X_A < q$;
Calculate $Y_A = \alpha^{X_A} \bmod q$
Y_A
User B
Generate random $X_B < q$;
Calculate $Y_B = \alpha^{X_B} \bmod q$;
Calculate $K = (Y_A)^{X_B} \bmod q$
Y_B
Calculate $K = (Y_B)^{X_A} \bmod q$

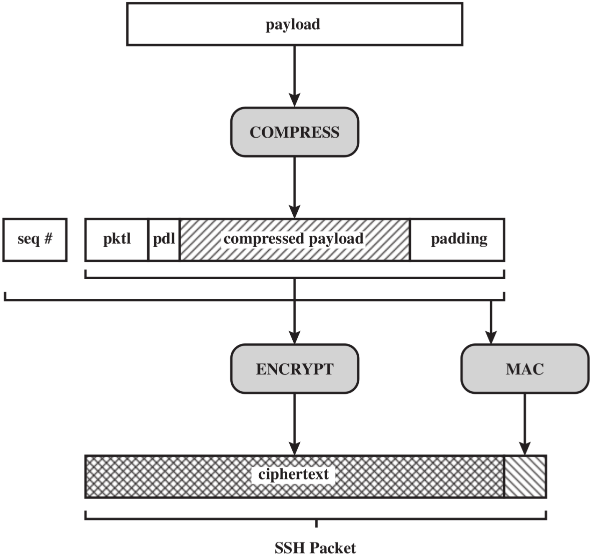
SSH Key Exchange with Diffie-Hellman

- SSH notation: $q = P$, $\alpha = G$, $Y_A = e$, $Y_B = f$
- ID string for client and server: ID_C, ID_S; SSH_MSG_KEXINIT message from client and server: M_C, M_S
- Server key pair: (PU_S, PR_S); assume client knows/trusts PU_S
- Client and server have agreed upon hash and encryption algorithms

SSH Key Exchange with Diffie-Hellman

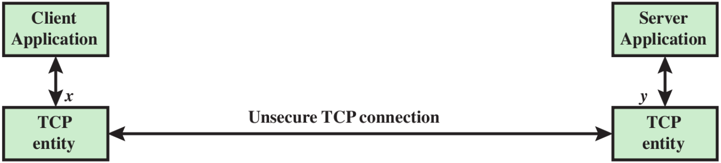
(see Wireshark capture)

SSH Transport Layer Packet Formation



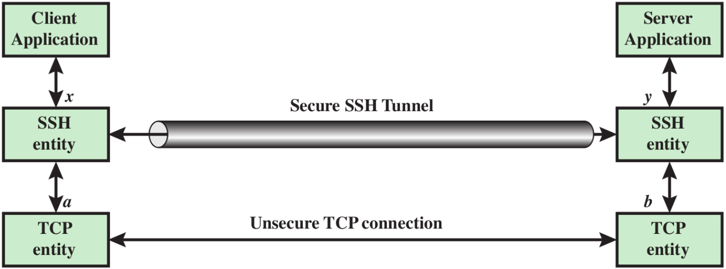
pktl = packet length
pdl = padding length

TCP Connection



a and *b* are application port numbers

SSH Tunnel over TCP Connection



x and *y* are application port numbers, *a* and *b* are port numbers used by SSH

SSH Tunnels

- Allow normal (unsecured) applications to securely transfer data
- Bypass firewalls by using different ports
- Local forwarding: traffic to local port is sent via SSH client to remote port
- Remote forwarding: traffic to remote port is sent via SSH server to local port