CNT4406/5412 Network Security
SSL/TLS

Zhi Wang

Florida State University

Fall 2014

Introduction

SSL/TLS runs on top of TCP and allows two parties to authenticate and securely communicate with each otherf

Introduction

SSL/TLS runs on top of TCP and allows two parties to authenticate and securely communicate with each otherf

- SSL was developed by Netscape, and runs on top of TCP

Introduction

SSL/TLS runs on top of TCP and allows two parties to authenticate and securely communicate with each otherf

- SSL was developed by Netscape, and runs on top of TCP
  ➠ SSLv2 was deployed in Netscape Navigator 1.1 in 1995

Introduction

SSL/TLS runs on top of TCP and allows two parties to authenticate and securely communicate with each otherf

- SSL was developed by Netscape, and runs on top of TCP
  - ➠ SSLv2 was deployed in Netscape Navigator 1.1 in 1995
  - ➠ SSLv3 was released in 1996 to address security flaws in SSLv2

Introduction

SSL/TLS runs on top of TCP and allows two parties to authenticate and securely communicate with each otherf

- SSL was developed by Netscape, and runs on top of TCP
  - ➠ SSLv2 was deployed in Netscape Navigator 1.1 in 1995
  - ➠ SSLv3 was released in 1996 to address security flaws in SSLv2
- IETF proposes TLS (Transport Layer Security) to standardize it

Introduction

SSL/TLS runs on top of TCP and allows two parties to authenticate and securely communicate with each otherf

- SSL was developed by Netscape, and runs on top of TCP
  - ➠ SSLv2 was deployed in Netscape Navigator 1.1 in 1995
  - ➠ SSLv3 was released in 1996 to address security flaws in SSLv2
- IETF proposes TLS (Transport Layer Security) to standardize it
  - ➠ v1.0: RFC2246, 1999; v1.1: RFC4346, 2006; v1.2: RFC5246, 2008

Introduction

SSL/TLS runs on top of TCP and allows two parties to authenticate and securely communicate with each otherf

- SSL was developed by Netscape, and runs on top of TCP
  - ➠ SSLv2 was deployed in Netscape Navigator 1.1 in 1995
  - ➠ SSLv3 was released in 1996 to address security flaws in SSLv2
- IETF proposes TLS (Transport Layer Security) to standardize it
  - ➠ v1.0: RFC2246, 1999; v1.1: RFC4346, 2006; v1.2: RFC5246, 2008
  - ➠ TLS for UDP (aka. DTLS) is defined in RFC 6347, 2012

Introduction...

- SSL relies on TCP for reliable communication (e.g., retransmission...)

Introduction...

- SSL relies on TCP for reliable communication (e.g., retransmission...)
  ➠ rogue packet problem: TCP accepts well-formated rogue packets, SSL discards them, but TCP won't accept data in the same range

Introduction...

- SSL relies on TCP for reliable communication (e.g., retransmission...)
  ➠ rogue packet problem: TCP accepts well-formated rogue packets, SSL discards them, but TCP won't accept data in the same range
- SSL/TLS is the de facto standard for web security

Introduction...

- SSL relies on TCP for reliable communication (e.g., retransmission...)
  ➠ rogue packet problem: TCP accepts well-formated rogue packets, SSL discards them, but TCP won't accept data in the same range
- SSL/TLS is the de facto standard for web security
  ➠ HTTPS uses TCP port 443

Introduction: SSL Service

- Peer authentication

Introduction: SSL Service

- Peer authentication
- Negotiation of security parameters

Introduction: SSL Service

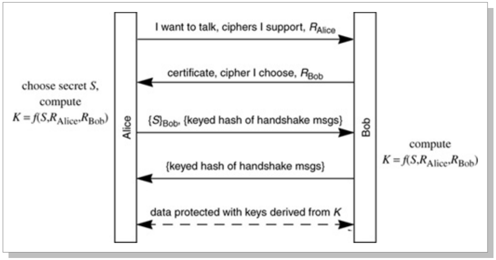
- Peer authentication
- Negotiation of security parameters
- Establishment of session keys

Introduction: SSL Service

- Peer authentication
- Negotiation of security parameters
- Establishment of session keys
- Data confidentiality and integrity

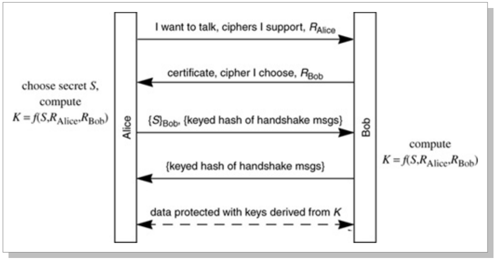
SSL Overview

- Message 1, 2 negotiate a cipher and exchange two random numbers


Alice
Bob
I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
$\{S\}_{Bob}$, {keyed hash of handshake msgs}
{keyed hash of handshake msgs}
data protected with keys derived from K
choose secret S, compute $K = f(S, R_{Alice}, R_{Bob})$
compute $K = f(S, R_{Alice}, R_{Bob})$

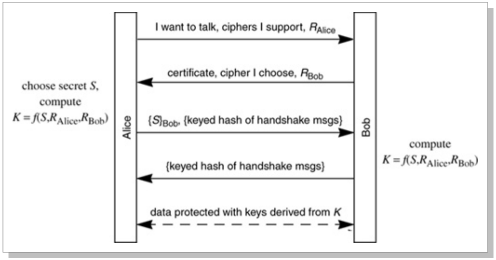
SSL Overview

- Message 1, 2 negotiate a cipher and exchange two random numbers
  ➠ R_{Alice} and R_{Bob} are combined with S to form keys


Alice
Bob
I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
choose secret S, compute $K = f(S,R_{Alice},R_{Bob})$
$\{S\}_{Bob}$, {keyed hash of handshake msgs}
{keyed hash of handshake msgs}
compute $K = f(S,R_{Alice},R_{Bob})$
data protected with keys derived from K

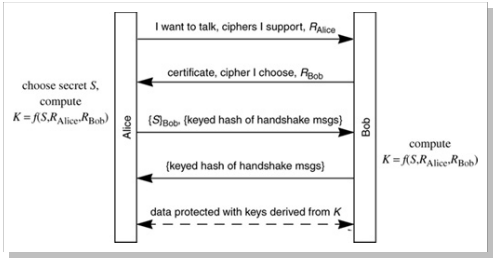
SSL Overview

- Message 1, 2 negotiate a cipher and exchange two random numbers
  - R_{Alice} and R_{Bob} are combined with S to form keys
  - normally, only the server provides a certificate


I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
choose secret S, compute $K = f(S,R_{\text{Alice}},R_{\text{Bob}})$
Alice
$\{S\}_{\text{Bob}}$, {keyed hash of handshake msgs}
Bob
compute $K = f(S,R_{\text{Alice}},R_{\text{Bob}})$
{keyed hash of handshake msgs}
data protected with keys derived from K

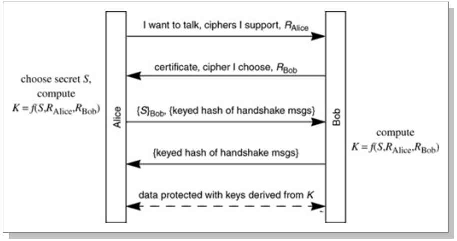
SSL Overview

- Message 1, 2 negotiate a cipher and exchange two random numbers
  ➠ R_{Alice} and R_{Bob} are combined with S to form keys
  ➠ normally, only the server provides a certificate
  ➠ to prevent MITM attack, user need to verify the certificate matches the web site (browers may give warning about it)


I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
$\{S\}_{Bob}$, {keyed hash of handshake msgs}
{keyed hash of handshake msgs}
data protected with keys derived from K
choose secret S, compute $K = f(S, R_{Alice}, R_{Bob})$
Alice
Bob
compute $K = f(S, R_{Alice}, R_{Bob})$

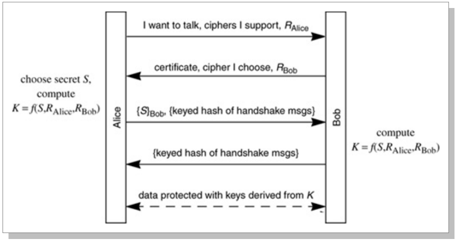
SSL Overview...

- Message 3, 4 establish keys and authenticate the handshake messages


Alice
Bob
I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
choose secret S, compute $K = f(S, R_{\text{Alice}}, R_{\text{Bob}})$
$\{S\}_{\text{Bob}}$, {keyed hash of handshake msgs}
compute $K = f(S, R_{\text{Alice}}, R_{\text{Bob}})$
{keyed hash of handshake msgs}
data protected with keys derived from K

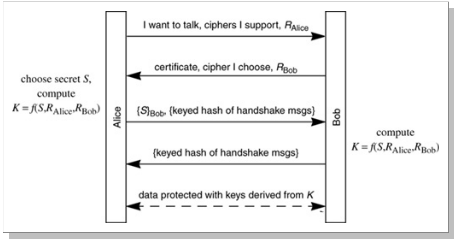
SSL Overview...

- Message 3, 4 establish keys and authenticate the handshake messages
  ➠ three pairs of keys are established for encryption, integrity, and IV


Alice
Bob
I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
$\{S\}_{Bob}$, {keyed hash of handshake msgs}
{keyed hash of handshake msgs}
data protected with keys derived from K
choose secret S, compute $K = f(S, R_{Alice}, R_{Bob})$
compute $K = f(S, R_{Alice}, R_{Bob})$

SSL Overview...

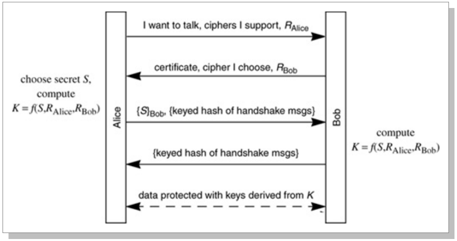
- Message 3, 4 establish keys and authenticate the handshake messages
  ➠ three pairs of keys are established for encryption, integrity, and IV
- Common practice for authentication:



Alice
Bob
I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
$\{S\}_{Bob}$, {keyed hash of handshake msgs}
{keyed hash of handshake msgs}
data protected with keys derived from K
choose secret S, compute $K = f(S, R_{Alice}, R_{Bob})$
compute $K = f(S, R_{Alice}, R_{Bob})$

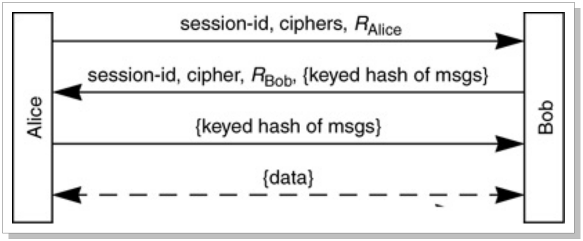
SSL Overview...

- Message 3, 4 establish keys and authenticate the handshake messages
  ➠ three pairs of keys are established for encryption, integrity, and IV
- Common practice for authentication:
  ➠ establish a secure channel through plaintext messages
  ➠ authenticate the previous messages to prevent MITM attacks


Alice
Bob
I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
choose secret S, compute $K = f(S, R_{Alice}, R_{Bob})$
$\{S\}_{Bob}$, {keyed hash of handshake msgs}
compute $K = f(S, R_{Alice}, R_{Bob})$
{keyed hash of handshake msgs}
data protected with keys derived from K

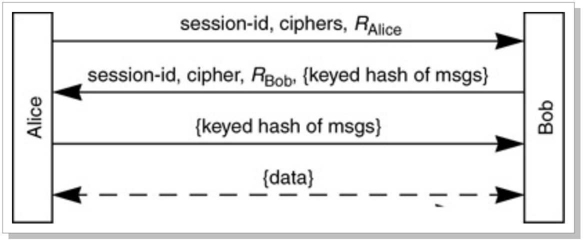
Sessions and Connections

- **SSL session** is a (long-lasting) association between the parties


Alice
Bob
session-id, ciphers, R_{Alice}
session-id, cipher, R_{Bob}, {keyed hash of msgs}
{keyed hash of msgs}
{data}

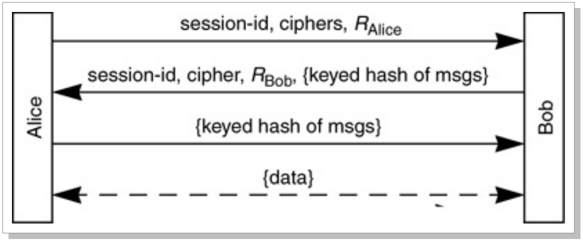
Sessions and Connections

- **SSL session** is a (long-lasting) association between the parties
  ➠ per-session master secret is established by public key cryptography


Alice
Bob
session-id, ciphers, R_{Alice}
session-id, cipher, R_{Bob}, {keyed hash of msgs}
{keyed hash of msgs}
{data}

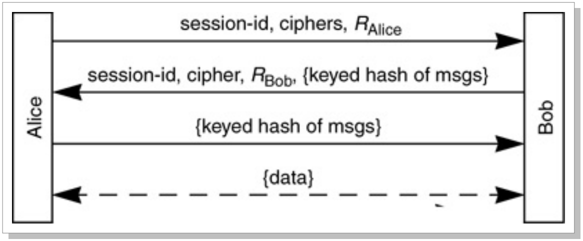
Sessions and Connections

- **SSL session** is a (long-lasting) association between the parties
  ➠ per-session master secret is established by public key cryptography
- **SSL connections** can be cheaply derived from the master secret


Alice
Bob
session-id, ciphers, R_{Alice}
session-id, cipher, R_{Bob}, {keyed hash of msgs}
{keyed hash of msgs}
{data}

Sessions and Connections

- **SSL session** is a (long-lasting) association between the parties
  ➠ per-session master secret is established by public key cryptography
- **SSL connections** can be cheaply derived from the master secret
  ➠ by doing a handshake that involves sending nonces


Alice
Bob
session-id, ciphers, R_{Alice}
session-id, cipher, R_{Bob}, {keyed hash of msgs}
{keyed hash of msgs}
{data}

Session Parameters

- Session ID
- X.509 public-key certificates
- Compression algorithm to use
- Cipher specifications

  ➠ encryption and message digest algorithms...
- Per-session master secret (48 bytes)

Connection Parameters

- Server and client nonces
- Three pairs of sever and client keys
  - ➠ encryption key and authentication key
  - ➠ initialization vectors
- Current message sequence number

SSL Messages

SSL messages are encoded into records, there are four record types

- Handshake $\rightarrow$establish a session key

SSL Messages

SSL messages are encoded into records, there are four record types

- Handshake →establish a session key
- Change cipher spec →start using the previously established keys

SSL Messages

SSL messages are encoded into records, there are four record types

- Handshake →establish a session key
- Change cipher spec →start using the previously established keys
- Application data →encrypted application data after the handshake

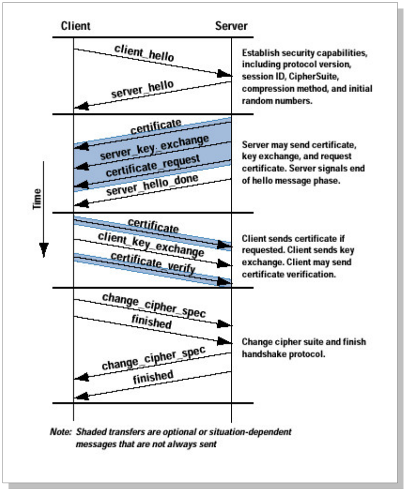
SSL Messages

SSL messages are encoded into records, there are four record types

- Handshake →establish a session key
- Change cipher spec →start using the previously established keys
- Application data →encrypted application data after the handshake
- Alert protocol→notification (warnings or fatal exceptions)

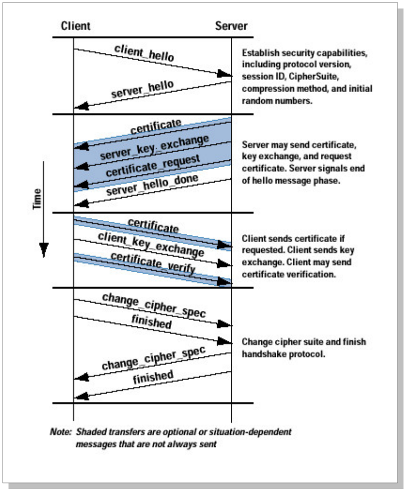
Handshake Messages

- Mandatory records:
  - client: `client_hello`, `client_key_exchange`, `change_cipher_spec`, `finished`


Client
Server
client_hello
server_hello
Establish security capabilities, including protocol version, session ID, CipherSuite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent

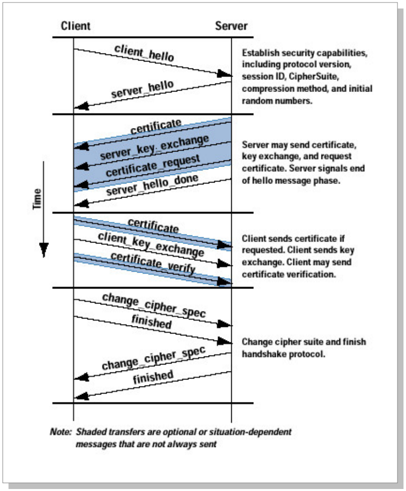
Handshake Messages

- Mandatory records:
  - client: client_hello, client_key_exchange, change_cipher_spec, finished
  - server: server_hello, server_hello_done, change_cipher_spec, finished


Client
Server
client_hello
server_hello
Establish security capabilities, including protocol version, session ID, CipherSuite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent

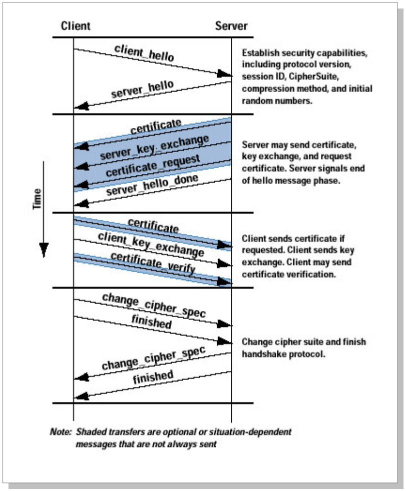
Handshake Messages

- Mandatory records:
  - client: `client_hello`, `client_key_exchange`, `change_cipher_spec`, `finished`
  - server: `server_hello`, `server_hello_done`, `change_cipher_spec`, `finished`
- Server almost always send a `certificate` record!


Client
Server
client_hello
server_hello
Establish security capabilities, including protocol version, session ID, CipherSuite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent

Handshake Messages

- Mandatory records:
  - client: `client_hello`, `client_key_exchange`, `change_cipher_spec`, `finished`
  - server: `server_hello`, `server_hello_done`, `change_cipher_spec`, `finished`
- Server almost always send a `certificate` record!


Client
Server
client_hello
server_hello
Establish security capabilities, including protocol version, session ID, CipherSuite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent

Handshake Messages: Client_Hello*

- The optional session_id allows session resumption

type = 1
length
version
random (R_{Alice})
length of session id (or 0 if absent)
session_id
length of cipher suites
cipher suites (each a 2-byte type)
length of compression list
compression methods

*the message formats are ordered as a normal SSL session

Handshake Messages: Server_Hello

- The optional session_id allows session resumption
- Bob's chosen cipher and compression method

type = 2
length
version
random (R_{Bob})
length of session id (or 0 if absent)
session_id
chosen cipher
chosen compression method

Handshake Messages: Certificate

- The server sends its certificate to the client
- The client may also send a certificate to the server if requested

type = 11
length
length (unnecessary field)
length of first certificate
first certificate
more pairs of certificates

Handshake Messages: Server_Hello_Done

- Server has finished sending its handshake messages

type = 14
length = 0

Handshake Messages: Client_Key_Exchange

- Client sends the pre-master secret encrypted the server's public key

type = 16
length
encrypted pre-master (S) secret

Handshake Messages: Change_Cipher_Spec

- All records following this will be protected with the negotiated ciphers

type = 20
version
length
ChangeCipherSpecType (set to 1)

Handshake Messages: Handshake Finished

- The message ensures the integrity of the exchange

type = 20
length (36 or 12)
digest of handshake messages

Handshake Messages: Certificate_Request

- Server requests client to send its certificate signed by selected CAs
  - ➠ it only list CA names, a CA can have more than one keys

type = 13
length
length of key type list
list of types of keys (e.g., RSA)
number of CA names
length of 1st CA names
1st CA name
more pairs of CA name length and name

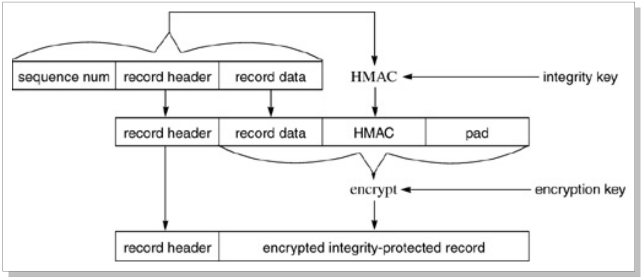
Handshake Messages: Certificate_Verify

- Client send it to prove it knows its private key

type = 15
length
length of signature
signature of the handshake message

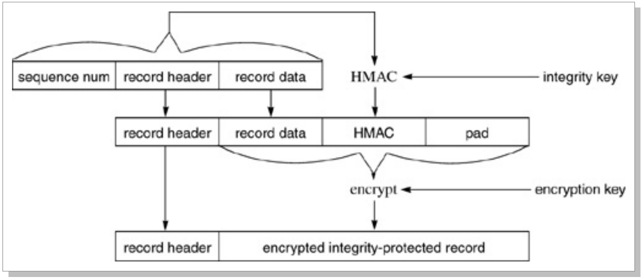
Application Data

- Application data are first fragmented into records
  - ➠ records are limited to 2^{14} bytes



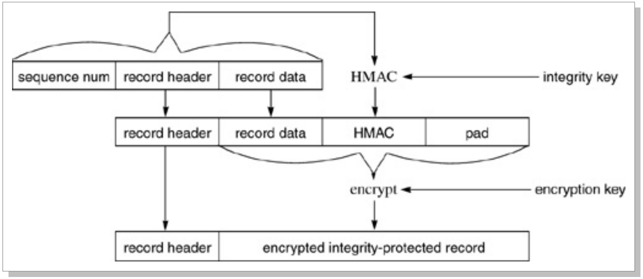
Application Data

- Application data are first fragmented into records
  ➠ records are limited to 2^{14} bytes
- Each record data is first compressed, then hashed with keyed HMAC

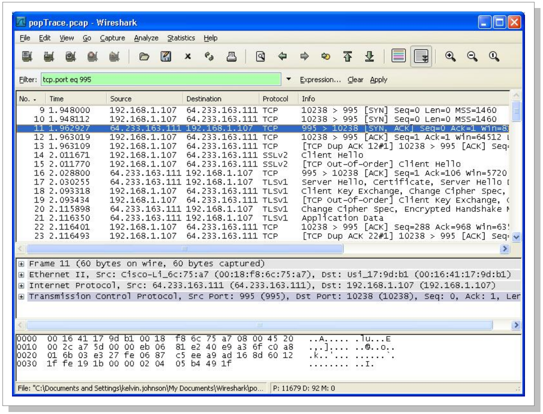


Application Data

- Application data are first fragmented into records
  - records are limited to 2^{14} bytes
- Each record data is first compressed, then hashed with keyed HMAC
- It is then encrypted and prepended with a record header



Demo: A Captured SSL Session


popTrace.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: tcp.port eq 995
Expression... Clear Apply

```
No. - Time      Source          Destination     Protocol  Info
   9 1.948000  192.168.1.107   64.233.163.111  TCP       10238 > 995 [SYN] Seq=0 Len=0 MSS=1460
  10 1.948112  192.168.1.107   64.233.163.111  TCP       10238 > 995 [SYN] Seq=0 Len=0 MSS=1460
  11 1.962927  64.233.163.111  192.168.1.107   TCP       995 > 10238 [SYN, ACK] Seq=0 Ack=1 Win=8
  12 1.963019  192.168.1.107   64.233.163.111  TCP       10238 > 995 [ACK] Seq=1 Ack=1 Win=64512 L
  13 1.963109  192.168.1.107   64.233.163.111  TCP       [TCP Dup ACK 12#1] 10238 > 995 [ACK] Seq=
  14 2.011671  192.168.1.107   64.233.163.111  SSLv2     Client Hello
  15 2.011770  192.168.1.107   64.233.163.111  SSLv2     [TCP Out-Of-Order] Client Hello
  16 2.028800  64.233.163.111  192.168.1.107   TCP       995 > 10238 [ACK] Seq=1 Ack=106 Win=5720
  17 2.030255  64.233.163.111  192.168.1.107   TLSv1     Server Hello, Certificate, Server Hello D
  18 2.093318  192.168.1.107   64.233.163.111  TLSv1     Client Key Exchange, Change Cipher Spec,
  19 2.093434  192.168.1.107   64.233.163.111  TLSv1     [TCP Out-Of-Order] Client Key Exchange, C
  20 2.115898  64.233.163.111  192.168.1.107   TLSv1     Change Cipher Spec, Encrypted Handshake M
  21 2.116350  64.233.163.111  192.168.1.107   TLSv1     Application Data
  22 2.116401  192.168.1.107   64.233.163.111  TCP       10238 > 995 [ACK] Seq=288 Ack=968 Win=63
  23 2.116493  192.168.1.107   64.233.163.111  TCP       [TCP Dup ACK 22#1] 10238 > 995 [ACK] Seq=

⊞ Frame 11 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: Cisco-Li_6c:75:a7 (00:18:f8:6c:75:a7), Dst: Usi_17:9d:b1 (00:16:41:17:9d:b1)
⊞ Internet Protocol, Src: 64.233.163.111 (64.233.163.111), Dst: 192.168.1.107 (192.168.1.107)
⊞ Transmission Control Protocol, Src Port: 995 (995), Dst Port: 10238 (10238), Seq: 0, Ack: 1, Len

0000  00 16 41 17 9d b1 00 18  f8 6c 75 a7 08 00 45 20   ..A..... .lu...E
0010  00 2c a7 5d 00 00 eb 06  81 e2 40 e9 a3 6f c0 a8   .,.].... ..@..o..
0020  01 6b 03 e3 27 fe 06 87  c5 ee a9 ad 16 8d 60 12   .k..'... ......`.
0030  1f fe 19 1b 00 00 02 04  05 b4 49 1f               ........ ..I.

File: "C:\Documents and Settings\kelvin.johnson\My Documents\Wireshark\po...   P: 11679 D: 92 M: 0
```

Summary

- SSL/TLS History
- SSL/TLS Overview
- SSL/TLS Details

- Next lecture: Web Security