Applied Cryptography and Network Security

Adam J. Lee
adamlee@cs.pitt.edu
6111 Sennott Square

Lecture #18: SSL
March 6, 2014

University of Pittsburgh

Announcements

Project Phase #3 due after break

Project Phase #4 will be posted after break

Overview

What is SSL/TLS and why do we need it?

How does SSL/TLS work?

- High-level intuitive explanation
- Packet-level details
- Key derivation
- Session resumption

PKI considerations when using SSL/TLS

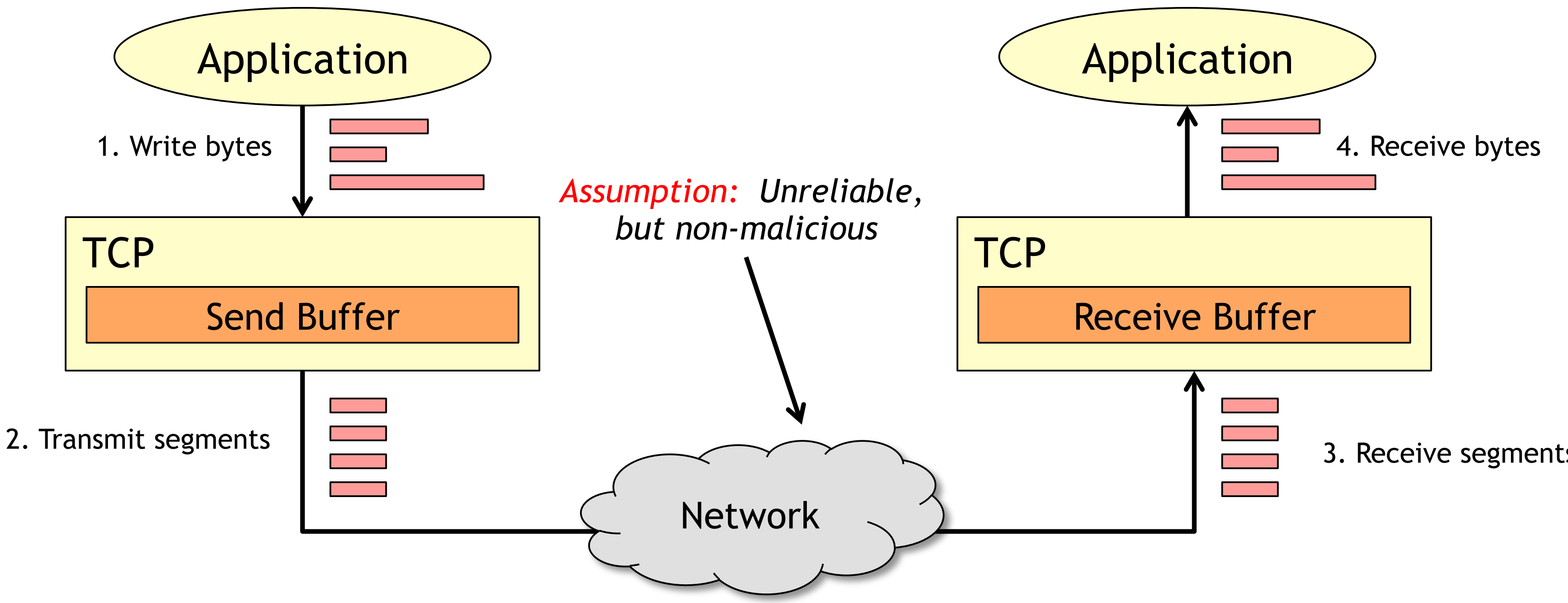
TCP is reliable, but not secure

TCP provides a reliable transport service

- Acknowledgements ensure that data is eventually delivered
- Checksums help protect packet integrity in the event of transient failure

Please note that:

- Active attackers can change the contents of packets
- Checksums are not cryptographic, so they can be forged



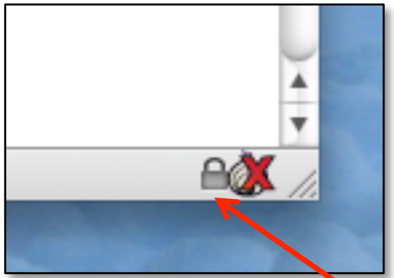
SSL and TLS were developed to provide applications with a secure means of utilizing TCP

Goal: Provide a generic means of authentication, confidentiality, and integrity protection to networked applications

That is, SSL/TLS were designed to simplify network security in the same way that Berkeley sockets simplified network programming

Where is SSL/TLS used?

- Web browsers
- Protecting FTP transmissions
- POP/IMAP over SSL
- VoIP security
- …



TLS protection in FireFox

Authentication in SSL can be one-way or mutual

- One way: Web browser authenticating, e.g., Amazon
- Mutual: B2B web services transactions

Historical Context

Building a protocol suite for secure networking applications is a solid idea. As such, there were many attempts made at this.

Secure Sockets Layer (SSL) was developed by Netscape

- Version 1 was never deployed
- Version 2 appeared in 1995
- Version 3 appeared in 1996

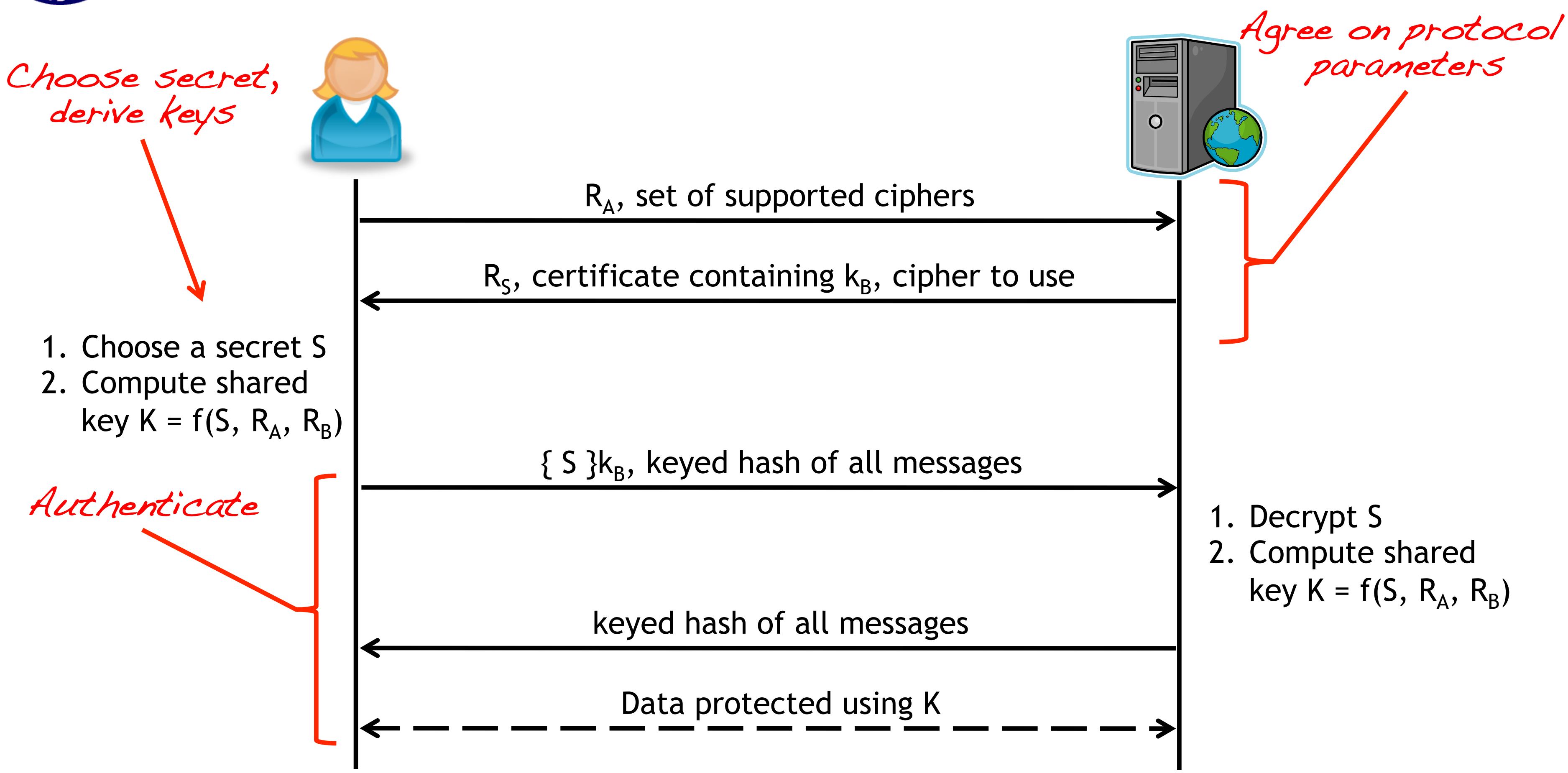
Microsoft developed PCT by tweaking SSL v2

- This was mostly a political move
- Forced Netscape to turn control of SSL over to the IETF

The IETF then developed the Transport Layer Security (TLS) protocol

- TLS 1.0 released in 1999 (RFC 2246)
- Most current version is TLS 1.2, which was released in 2008 (RFC 5246)

Viewed abstractly, TLS is actually a very easy to understand protocol



What is a cipher suite?

A cipher suite is a complete set of algorithmic parameters specifying exactly how the protocol will run

- Specified using a 16-bit identifier
- Also given a pseudo-meaningful name

Symmetric key cryptography will be done using AES

Example: `TLS_RSA_WITH_AES_128_CBC_SHA`

SHA-1 will be used to hash

This cipher suite is defined in the TLS specification

Public key operations will be carried out using RSA

AES will use a 128-bit key

AES will operate in CBC mode

In SSL/TLS, the client proposes a list of supported cipher specifications, and the server gets to choose which one will be used

Discussion

Why are cipher suites a good idea?

SSL and TLS transmit data one record at a time

SSL/TLS defines four specific message types

- Handshake records contain connection establishment/setup information
- The ChangeCipherSpec record is a flag used to indicate when cryptographic changes will go into effect
- Alert records contain error messages or other notifications
- Application_Data records are used to transport protected data

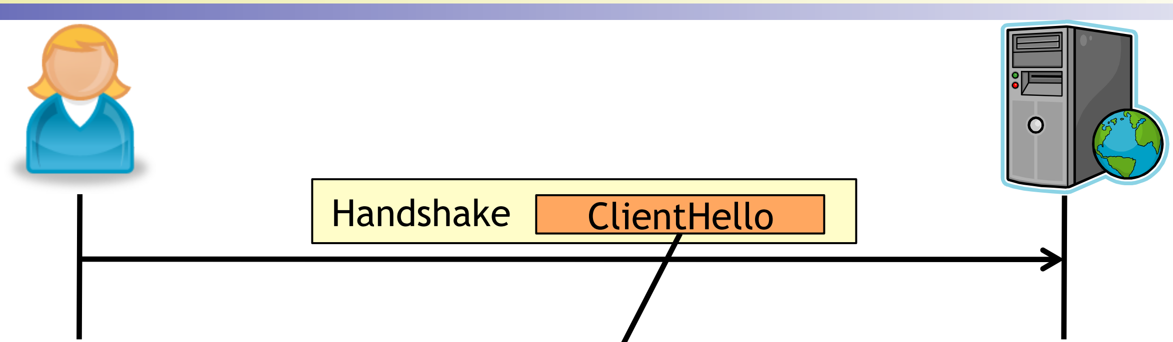
Note that a single record may contain multiple messages inside of it

Each record is delineated using a 5 byte record header

Bytes	***Content***
1	Record type code
2	Version number
2	Length

To illustrate this record/message sending process, let's look into the details of the protocol…

Handshake: Message 1

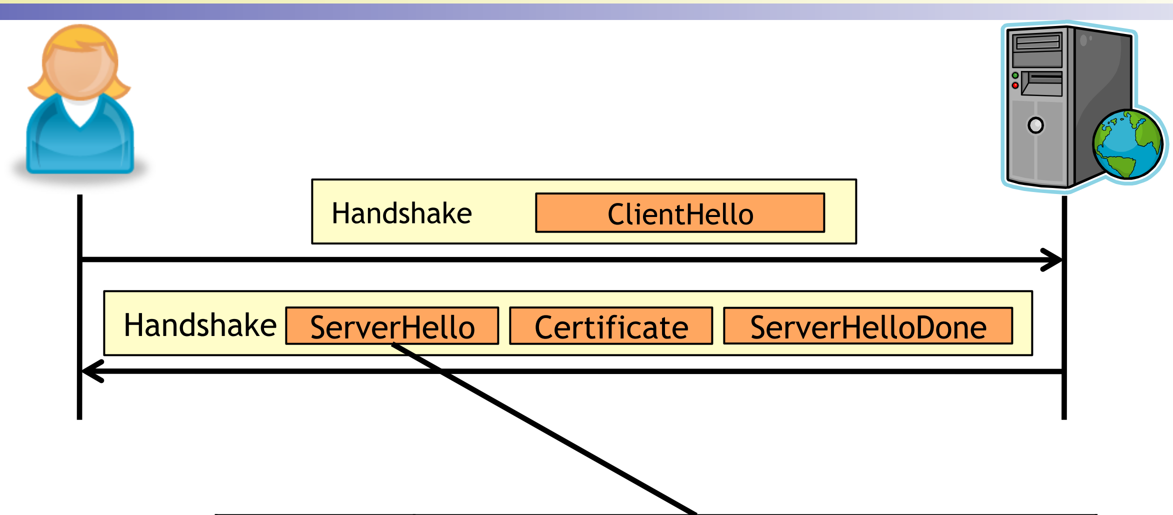


Bytes	*Content*
1	Message type: ClientHello
3	Length
2	Version number
32	R_A
1	Length of session ID
var	Session ID
2	Length of cipher suite list
var	List of supported cipher suites
1	Length of compression mode list
var	List of supported compression modes

Used to resume sessions. More on this later.

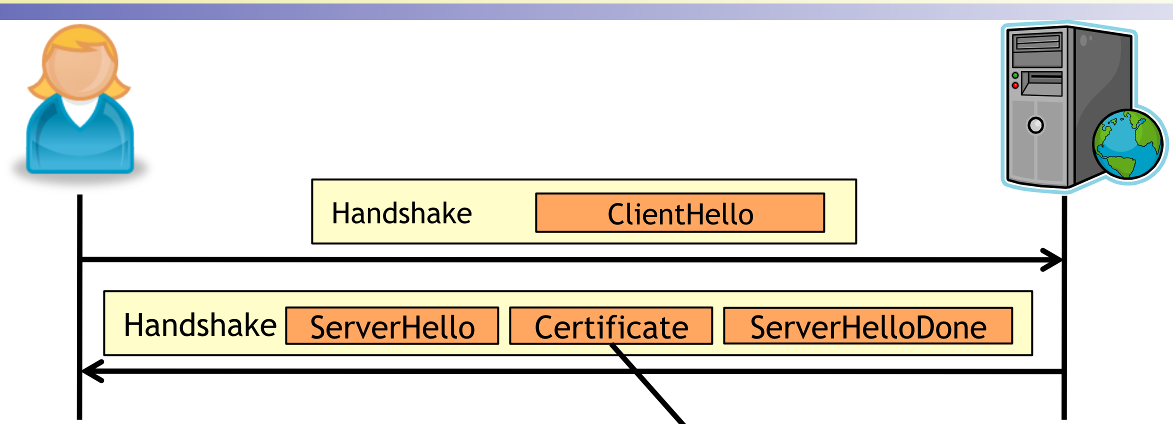
For efficiency reasons, SSL/TLS compresses the data that it transmits.

Handshake: Message 2



Bytes	***Content***
1	Message type: ServerHello
3	Length
2	Version number
32	R_S
1	Length of session ID
var	Session ID
2	Chosen cipher suite
1	Chosen compression method

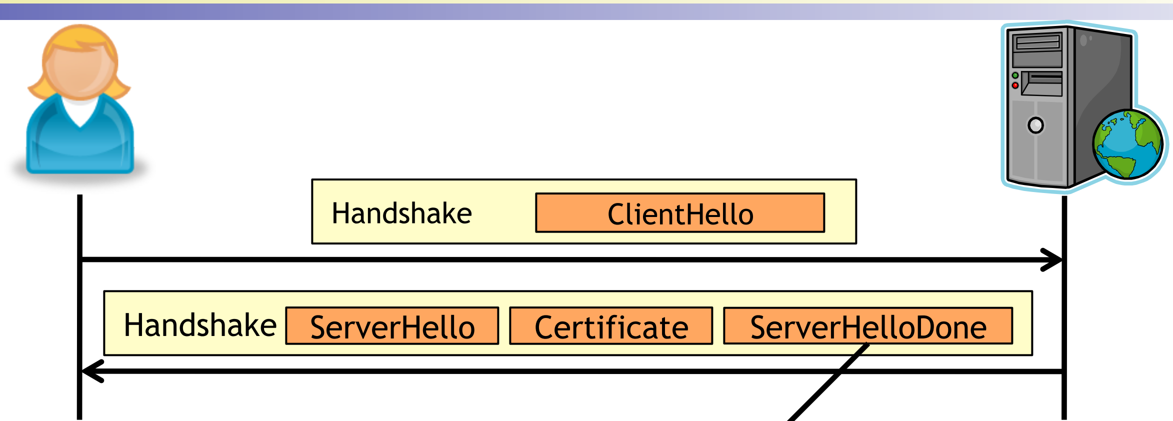
Handshake: Message 2



Bytes	*Content*
1	Message type: ServerHello
3	Length
3	(Redundant) length
3	Length of first certificate
var	First certificate
var	More (length, certificate) pairs

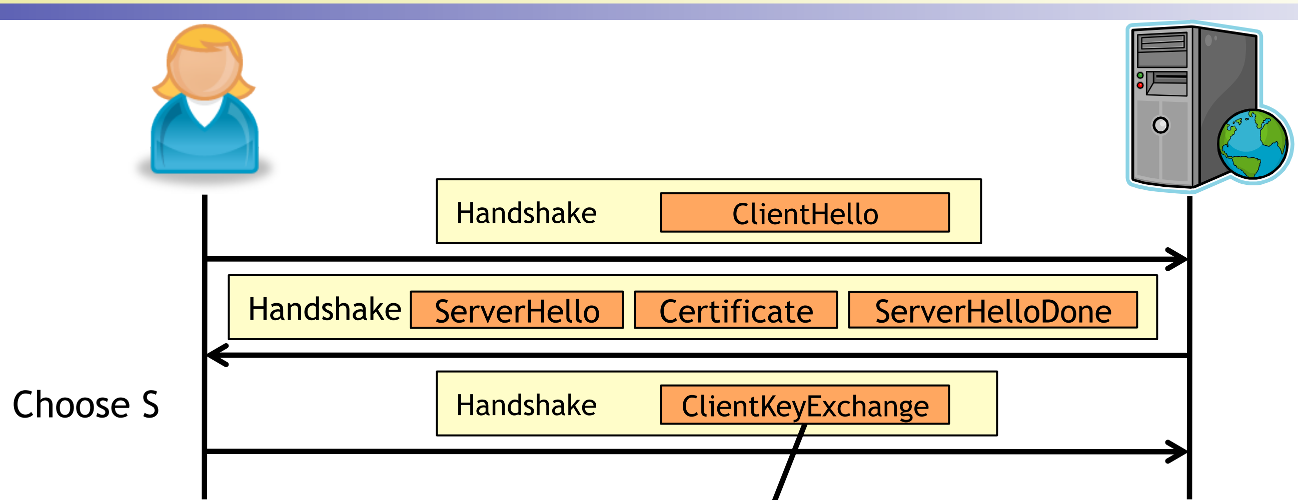
Why?

Handshake: Message 2



Bytes	***Content***
1	Message type: ServerHelloDone
3	Length = 0

Handshake: Message 3



Bytes	*Content*
1	Message type: ClientKeyExchange
3	Length = 0
2	(Redundant) length
var	Encrypted pre-master secret $\{ S \}k_S$

How are keys computed?

How are session keys computed?

RFC 5246 defines a pseudorandom function that is used to expand the master secret into an randomized key stream:

```
PRF(secret, seed) =
   HMAC(secret, A(1) || seed) ||
   HMAC(secret, A(2) || seed) ||
   HMAC(secret, A(3) || seed) || ...
```

The sequence A is defined as follows:

- A(0) = seed
- A(i) = HMAC(secret, A(i-1))

Note: The version of HMAC that is used depends on the cipher suite!

- E.g., `TLS_RSA_WITH_AES_128_CBC_SHA` uses HMAC-SHA-1

Key derivation, continued

First, the master secret is computed:

- master_secret = PRF(pre_master_secret, "master secret" $||$ R_A $||$ R_S)[0..47]

Next, a stream of random bytes is generated from the master secret

- key_block = PRF(master_secret, "key expansion" $||$ R_S $||$ R_A)

Keys are taken from the above block of key material in the following order

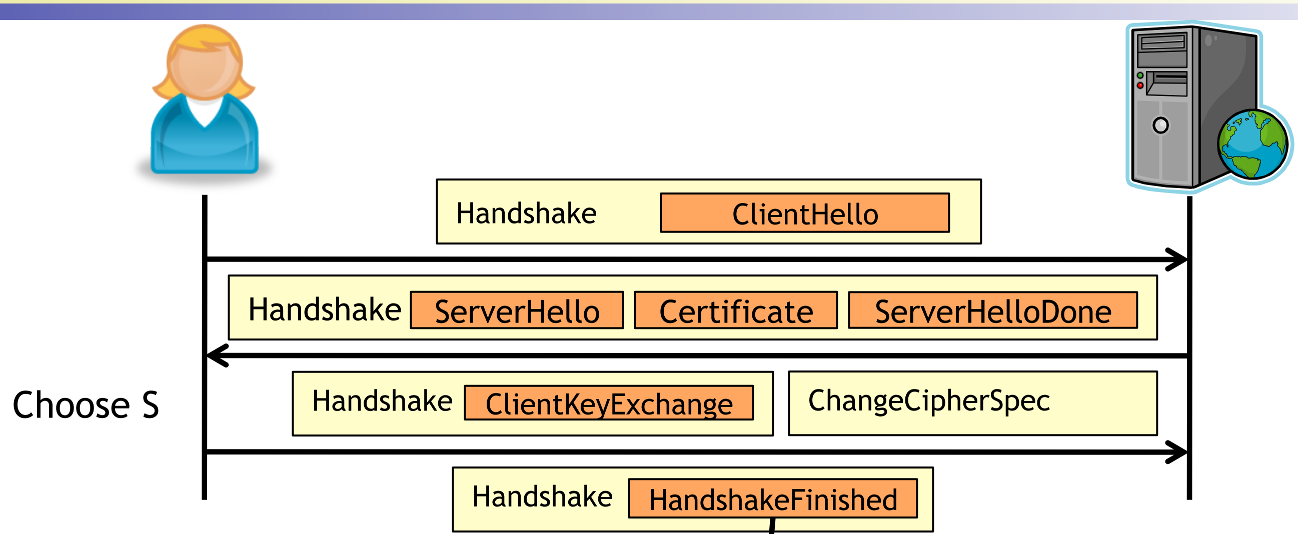
- Client → server MAC key
- Server → client MAC key
- Client → server symmetric encryption key
- Server → client symmetric encryption key
- Client → server IV
- Server → client IV

This process is called key expansion. Note that the amount of key material generated depends on the cipher suite chosen. (Why?)

Discussion

In TLS, the client provides the pre master secret, S, to the server. Why are R_A and R_S included when computing the master secret from the pre master secret?

Finishing the Handshake



Bytes	*Content*
1	Message type: HandshakeFinished
3	Length of digest
var	Keyed digest of message exchange

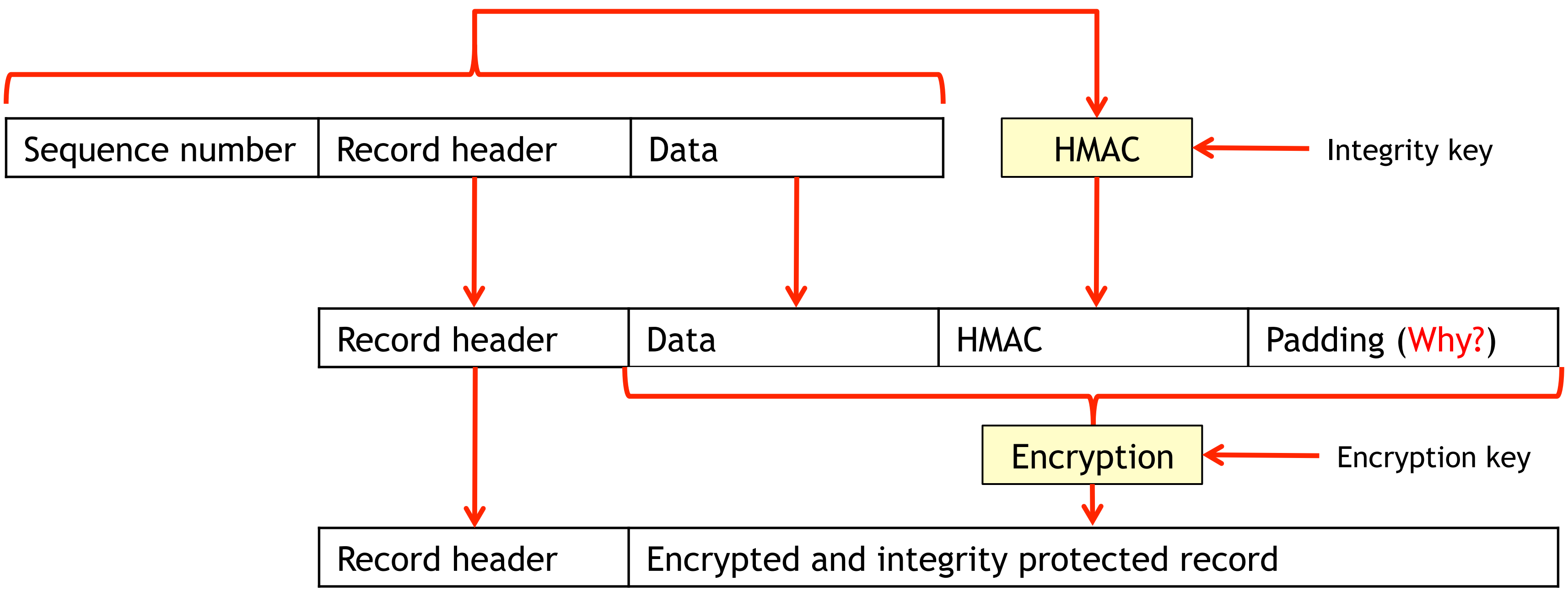
All records sent after ChangeCipherSuite are encrypted and integrity protected

All of this protection is afforded by the algorithms identified in the cipher suite chosen by the server

Example: `TLS_RSA_WITH_AES_128_CBC_SHA`

- Encryption provided using 128 bit AES in CBC mode
- Integrity protection provided by HMAC-SHA-1

Note: Data is protected one record at a time



Protocol summary

At a high level, this protocol proceeds in four phases

- Setup and parameter negotiation
- Key exchange/derivation
- Authentication
- Data transmission

For security reasons, both parties participate in (almost) all phases

- Setup: Client proposes, server chooses
- Key derivation: Randomness contributed by both parties
- Authentication: Usually, just the server is authenticated (Why?)
- Data transmission: Both parties can encrypt and integrity check

The low level details are not much more complicated than that!

This handshake procedure is fairly heavyweight

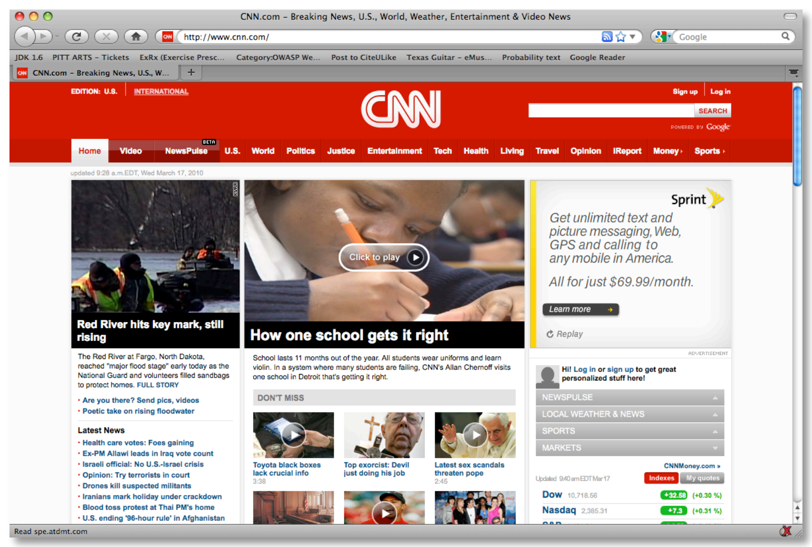
Public key cryptography is used by both parties

- Alice encrypts her pre-master secret using the server's public key
- The server decrypts this pre-master secret

So what! Aren't connections long lived?

Example: Visiting http://www.cnn.com

Visiting this single web page triggers over 130 separate HTTP connections!

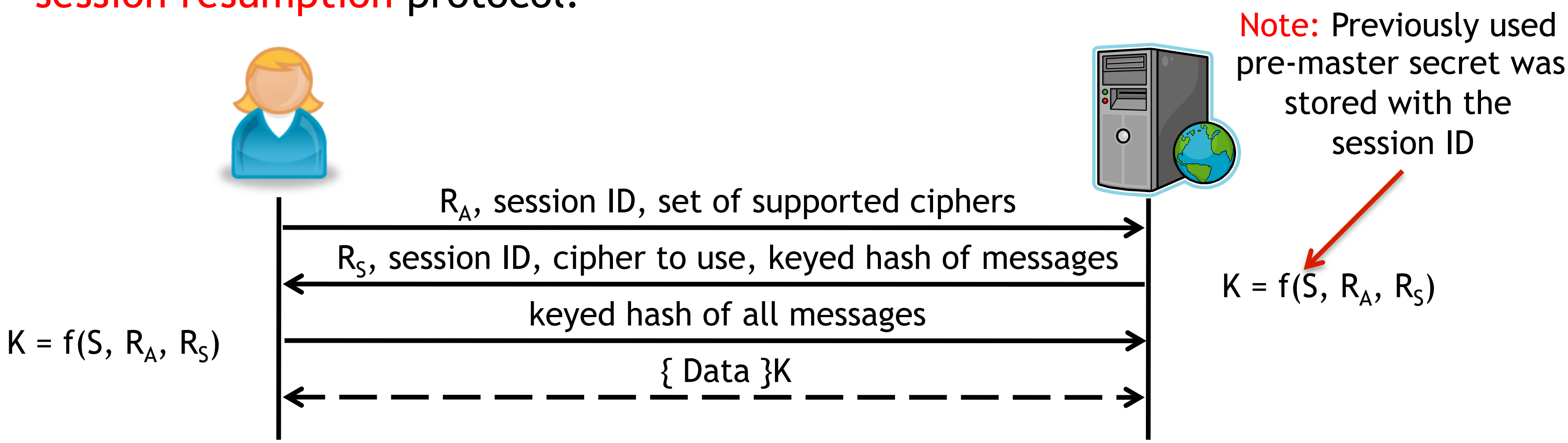


This is less than optimal.
Can we do better?

Sessions vs. Connections

In SSL/TLS, sessions are assumed to be long-lived entities that may involve many smaller connections

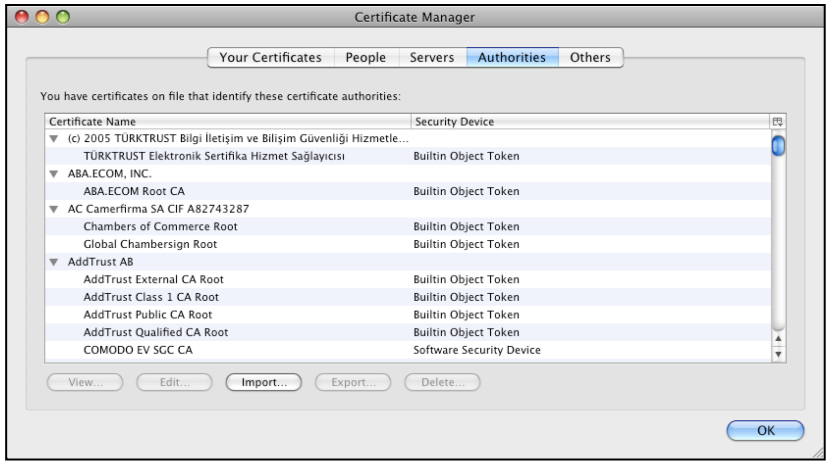
Connections can be spawned from an existing session using a streamlined session resumption protocol:



In this model, a single session could be set up with CNN and connections can be spawned as needed to retrieve content

Servers authenticate themselves using certificates. How do we authenticate the certificates?

Most SSL/TLS deployments use an oligarchy PKI model



That is, as long as the server presents a certificate chain that uses one of our trusted roots, we're happy

What about naming?

- Servers are usually known by their DNS names
- X.509 is not set up for DNS naming
- Usually the CN field of the X.509 certificate contains the DNS name

Example: C = US, ST = Washington, L = Seattle, O = Amazon.com Inc.,
CN = www.amazon.com

Why is this safe?

Conclusions

Although TCP provides a reliable data transfer protocol, it is not secure

- TCP can recover from bit-flips and dropped packets
- But malicious adversaries can alter data undectected

SSL/TLS provides cryptographic confidentiality and integrity protection for data sent over TCP

The security afforded by SSL/TLS is defined by using cipher suites

- Developers to easily incorporate new algorithms
- Security professionals can tune the level of security offered
- Breaking a cipher does not break the protocol

Next time: Exam discussion